VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Aplikace pro automatické pořizování snímků zájmových bodů

Android Application for Point of Interests Image Acquisition

2016 Lukáš Magnusek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lukáš Magnusek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Aplikace pro automatické pořizování snímků zájmových bodů
Android Application for Point of Interests Image Acquisition

Jazyk vypracování: čeština

Zásady pro vypracování:

Navrhněte a implementujte aplikaci umožňující automatické pořizování snímků z mobilního zařízení na základě jeho polohy nebo vnější události (prudká změna přetížení, přetočení apod.). Klientská aplikace poběží na mobilním zařízení s OS Android a bude komunikovat se serverem, kde bude zasílat pořízené snímky a stahovat aktuální nastavení. Vzhledem k tomu, že aplikace pořídí více snímků v dané lokalitě, měla by být schopna samostatně rozhodnout, který snímek je nejlepší (odfiltrovat rozmazané snímky, snímky oblohy apod.) Serverová komponenta umožní přehledné zobrazení pořízených dat (zobrazení na mapovém podkladu, procházení pořízených galerií apod.) Předpokládá se, že server bude podporovat více souběžně připojených klientů.

1. Implementujte klientskou stranu aplikace pro OS Android.
2. Navrhněte vhodný algoritmus z oblasti zpracování obrazu pro výběr "nejlepšího" snímku ze série pořízených.
3. Navrhněte vhodný způsob bufferování snímků před samotným odesláním na server.
4. Implementujte zabezpečený přenos snímků a konfigurací mezi klientem a serverem.
5. Vytvořte vhodné webové rozhraní pro konfiguraci a prohlížení pořízených dat.

Seznam doporučené odborné literatury:

[1] Rafael C. Gonzalez, Richard E. Woods, Digital Image Processing, Prentice Hall; 3 edition, 2007, ISBN 978-0131687288
[2] Reto Meier, Professional Android 4 Application Development, Wrox, 3 edition, 2012, ISBN 978-1118102275
[3] Gred Nudelman, Android Design Patterns: Interaction Design Solutions for Developers, Wiley, 2013, ISBN 978-1118394151

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Ing. Michal Krumnikl, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 26. dubna 2016 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 26. dubna 2016 .....

Rád bych na tomto místě poděkoval panu Mgr. Ing. Michalu Krumniklovi, Ph.D., za jeho rady, návrhy a odbornou pomoc při vytváření této diplomové práce.

Abstrakt

Cílem této diplomové práce je navrhnout a naimplementovat aplikaci pro operační systém Android, která automaticky pořizuje fotky zájmových bodů na základě polohy tohoto zařízení, nebo jeho zatřesením. Hlavním úkolem aplikace je vyfotit určitý počet snímků v dané lokalitě a pomocí vybraného algoritmu z oblasti zpracování obrazu vybrat nejlepší snímek ze série pořízených. Všechny tyto snímky se budou ukládat v zařízení a budou přehledně zobrazeny v galerii aplikace.

Dále je úkolem vytvořit webový server, kde bude klientská aplikace pro Android zasílat nejlepší vyfocené snímky spolu s informacemi, kde byl snímek pořízen, kdy byl pořízen apod. Toto spojení by mělo být zabezpečené před odposloucháváním a podvržením dat. Na webovém serveru si bude uživatel moct přidávat nové lokace, které si bude následně stahovat klientská aplikace. Na serveru se také předpokládá zobrazení zaslaných snímků spolu s doplňujícími informacemi těchto snímků.

Klíčová slova: Android, mobilní aplikace, webový server, zájmové body, obraz, hodnocení kvality obrazu, IQA

Abstract

The main goal of this thesis is to design and implement an application for the Android operating system, which automatically captures photos of points of interest based on the position of the device, or by shaking with the device. The main task of the application is to take pictures in the location and by algorithmic means select the best image from the sequence of images taken during the approach. All these images will be stored in the device and will be displayed in the gallery of the application.

The next task is to create a web server, where the client application for Android will be sending the best photos with information, where the picture was taken, when it was taken etc. This connection should be secure against eavesdropping and data spoofing. On the web server, a user will be able to add new locations, which will be then downloaded by the client application. The server will also provide an opportunity to display the images with the additional information uploaded by the client.

Key Words: Android, mobile application, web server, points of interest, image, image quality assessment, IQA

Obsah

Seznam použitých zkratek a symbolů

1D	–	One-Dimensional
2D	–	Two-Dimensional
API	–	Application Programming Interface
ASP.NET	–	Active Server Pages .NET
BIQI	–	Blind Image Quality Index
BRISQUE	–	Blind/Referencless Image Spatial QUality Evaluator
BSD	–	Berkeley Software Distribution
BT	–	Broadcasting Service
CPU	–	Central Processing Unit
DCT	–	Discrete Cosine Transform
DIIVINE	–	Distortion Identification-based Image Verity and INtegrity Evaluation
DIS	–	Distorted Image Statistics
DS	–	Double Stimulus
DSCQS	–	Double Stimulus Continual Quality Scale
DSIS	–	Double Stimulus Impairment Scale
DWT	–	Discrete Wavelet Transform
FF	–	Fast Fading
FR	–	Full Reference
GB	–	Gaussian Blur
GGD	–	Generalized Gaussian Distribution
GNU	–	Gnu's Not Unix
GPS	–	Global Positioning System
HTTP	–	HyperText Transfer Protocol
HTTPS	–	HyperText Transfer Protocol Secure
HVS	–	Human Visual System
HW	–	Hardware
ID	–	Identification
IDE	–	Integrated Development Environment
IIS	–	Internet Information Services
IP	–	Internet Protocol
IQA	–	Image Quality Assessment
ITU-R	–	International Telecommunication Union - Radiocommunication
JP2K	–	JPEG2000
JPEG	–	Joint Photographic Experts Group
JPEG2000	–	Joint Photographic Experts Group 2000

JSON	–	JavaScript Object Notation
LG	–	Lucky Goldstar
LGPL	–	Lesser General Public License
LIVE	–	Laboratory for Image & Video Engineering
MOS	–	Mean Opinion Score
MPEG	–	Motion Picture Experts Group
MSE	–	Mean Squared Error
NIQE	–	Naturalness Image Quality Evaluator
NR	–	No Reference
NSS	–	Natural Scene Statistics
NT	–	New Technology
ORM	–	Object Relational Mapping
OS	–	Operating System
PC	–	Personal Computer
PPI	–	Pixels Per Inch
PSNR	–	Peak Signal-to-Noise Ratio
RGB	–	Red, Green, Blue
RR	–	Reduced Reference
SDK	–	Software Development Kit
SHA	–	Secure Hash Algorithm
SQL	–	Structured Query Language
SŘBD	–	Systém Řízení Báze Dat
SS	–	Single Stimulus
SSCQE	–	Single Stimulus Continuous Quality Evaluation
SSIM	–	Structural Similarity Index
SSL	–	Secure Sockets Layer
SSM	–	Single Stimulus Method
SVM	–	Support Vector Machines
SW	–	Software
TLS	–	Transport Layer Security
URL	–	Uniform Resource Locator
Wi-Fi	–	Wireless Fidelity
WN	–	White Noise
WS	–	Web Service

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V této diplomové práci shrnuji poznatky, které jsem se za léta studií na Vysoké škole báňské naučil a osvojil si principy programování v různých programovacích jazycích. Téma této práce jsem si vybral, jelikož programování pro operační systém Android v jazyce Java je mi velice blízké a chtěl bych se mu v budoucnu věnovat. I přesto, že jsem si toto téma zvolil převážně kvůli mé oblibě systému Android, lákalo mě využití dalších jazyků, jako jsou JavaScript a C#, resp. ASP.NET, které jsou v diplomové práci také použity. Diplomová práce dostala název *Aplikace pro automatické pořizování snímků zájmových bodů*. Ve stručnosti jde o aplikaci, která automaticky pořizuje snímky v závislosti na poloze zařízení, případně jeho zatřesením.

Jak již bylo naznačeno, úkolem této diplomové práce bylo v prvé řadě navrhnout a implementovat klientskou stranu aplikace pro operační systém Android. Tato aplikace by umožňovala registraci a přihlášení uživatele, aby mohl používat jak klientskou aplikaci, tak webové rozhraní na serveru, kde jsou tyto přihlašovací údaje uloženy v databázi. Po přihlášení uživatele do aplikace, může tento uživatel začít snímat série fotek, a to způsobem, že buďto se zařízením zatřese, nebo se přiblíží nastavené lokaci (za předpokladu, že si tuto pozici nastavil ve webovém rozhraní na serveru). Po spuštění jedné z těchto událostí, začne aplikace automaticky fotit fotky. Aplikace si po určitou dobu ukládá všechny fotky do mezipaměti a po vyfocení nastaveného počtu snímků, nebo vzdálení se od nastavených lokací, automaticky objektivně vybere nejlepší snímek, se snahou aby co nejpřesněji odpovídal subjektivnímu hodnocení člověka. Po vyhodnocení snímků aplikace automaticky pošle přes zabezpečené spojení nejlepší snímek na server spolu s informacemi o pořízení snímku a lokaci. Aplikace si rovněž ukládá všechny vyfocené snímky dané pozice, které se spolu s informacemi o fotografii a pozici zobrazují v galerii aplikace. Neméně podstatným úkolem bylo vytvoření webového rozhraní serveru, kde se uživatel přihlásí a má možnost navolení lokací, na kterých chce fotit a zobrazit si galerii pořízených dat. Pro implementaci tohoto serveru jsem si vybral framework ASP.NET.

Ovšem velmi podstatným úkolem této diplomové práce je objektivní hodnocení kvality obrazu bez referenčního snímku. Aplikace musí umět sama rozhodnout, který snímek ze série pořízených je nejlepší s ohledem na lidské vnímání kvality obrazu. Toto hodnocení kvality obrazu strojem je i v dnešní době bohužel poměrně těžkým úkolem.

V následujících kapitolách tedy nejprve uvedu potřebnou teorii, poté možnosti posuzování kvality obrazu, jejich matematické výpočty a jaký algoritmus pro posouzení kvality obrazu jsem si pro aplikaci vybral a proč. Následně provedu analýzu a návrh aplikací (klient i server). Dále popíšu jejich fungování, a jak jsem samotné aplikace implementoval. Na závěr uvedu testování obou aplikací a dosažené výsledky práce.

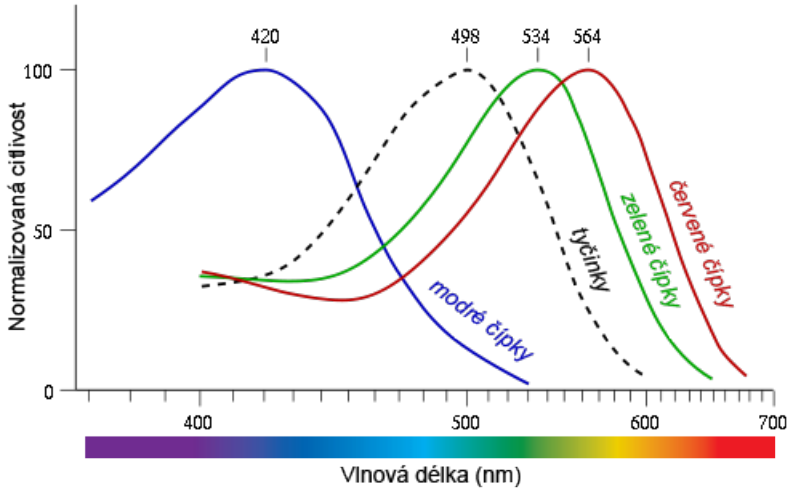
2 Hodnocení kvality obrazu

Hodnocení kvality obrazu neboli posuzování kvality obrazu (IQA) je v dnešní době velmi klíčovým tématem v oblasti zpracování obrazu. Díky obrovskému rozvoji za poslední léta v této oblasti, je velká poptávka po algoritmech, které by samostatně rozhodovaly o kvalitě obrázku z hlediska systému lidského vnímání (HVS). Je tedy snaha převést psycho-fyzikální vlastnosti HVS do objektivního modelu, který v ideálním případě na 100% koreluje s těmito vlastnostmi. Bohužel těchto algoritmů je v dnešní době velmi málo a nejsou příliš přesné. Lze tedy usoudit, že se jedná o velmi těžkou a problematickou disciplínu.

2.1 Teoretické základy pro lidský zrakový systém

Abychom mohli vytvořit určitý objektivní model pro posuzování kvality obrazu, je třeba nejprve pochopit základy fungování zrakového systému člověka. Ve zkratce jde o to, že je potřeba v digitálním obraze najít informace, které co nejlépe odpovídají lidskému vnímaní, tedy HVS. Tyto informace je třeba následně z obrazu extrahovat a provádět nad nimi testy. [1].

Aby si lidské oko mohlo vytvořit zrakový vjem je potřeba světlo dopadající na sítnici, která obsahuje 2 druhy fotoreceptorů (tyčinky a čípky). Lidské oko obsahuje 3 druhy čípku, a to pro: modrofialovou (cca 425 nm), zelenou (cca 530 nm) a oranžovou, resp. červenou (cca 560 nm) barvu. Jak lze vidět na obrázku 1, je lidské oko nejcitlivější právě na těchto třech vlnových délkách. Lze tedy vidět jistou souvislost s barevnými modely v počítačové grafice (konkrétně s modelem RGB). Čípky jsou tedy barevně citlivé receptory, které zajišťují tzv. fotopické vidění a tyčinky jsou pouze jasově citlivé receptory, které zajišťují tzv. skotopické vidění.



Obrázek 1: Citlivost lidského oka na barvy

Jelikož tyto čípky a tyčinky nejsou na sítnici rovnoměrně rozloženy, vyplývá z toho tedy, že lidský zrakový systém (HVS) není schopen vnímat celý obraz v rovnoměrném rozlišení.

Při modelování HVS se tedy klade největší důraz právě na fotopické vidění a nebere se ohled na skotopické vidění. Předpokládá se tedy, že světelné podmínky na zkoumaném obraze budou normální. Tato problematika modelování HVS je daleko složitější a byla by nad rámec této diplomové práce. Detailnější popis lze najít např. v [2].

2.2 Reprezentace digitálního rastrového obrazu

Z obrázku 1 tedy vidíme, že lidské oko je schopné vidět část elektromagnetického spektra o vlnových délkách přibližně 380 nm až 750 nm. Této části spektra říkáme barevné spektrum. Toto spektrum barev potřebujeme převést do digitální podoby. Pokud budeme mluvit pouze o rastrové grafice, jde tedy v základu o převod analogové formy obrazu (obraz viditelný lidským okem) do formy digitální (obraz v digitální podobě). Tento digitální obraz je uložen v paměti v pixelech (složením slov picture a element), což jsou nejmenší, nedělitelné jednotky obrazové informace. Pixely jsou uspořádány v mřížce, která udává rozlišení digitálního obrazu. Pokud bychom měli černobílý obraz, pixel by nabýval pouze hodnot 0 a 1, pokud bychom měli obraz v odstínech šedi nabýval by hodnot 0 - 255 a pokud bychom měli barevný obraz, byl by pixel dán aditivním barevným modelem RGB. Jak již bylo řečeno, tento model je založen na faktu, že lidské oko je citlivé na tyto 3 barvy (modrou, zelenou a červenou).

Zde již vzniká problém jak objektivně určit kvalitu obrazu. Existují jednoduché algoritmy, které se tuto hodnotu (neboli index kvality) snaží určit v prostorové doméně, ale i složitější algoritmy, které pracují ve spektrální doméně a využívají diskrétní kosinové nebo diskrétní vlnkové transformace. Těmito postupy se ale budeme zabývat později.

2.3 Subjektivní a objektivní hodnocení kvality obrazu

U hodnocení kvality obrazu rozlišujeme dva důležité pojmy: subjektivní a objektivní hodnocení. U subjektivního hodnocení je zapotřebí určitého počtu osob (testovacích subjektů), kteří hodnotí kvalitu obrazu. Tato metoda je velmi náročná na čas a je potřeba velký počet testovacích subjektů. Z důvodu těchto nevýhod se zavádějí objektivní metody měření, které používají matematických výpočtů. Tyto analyzátory mohou být založeny na hardwarovém, softwarovém nebo kombinovaném principu. V této práci se budeme samozřejmě bavit pouze o softwarovém principu. Tyto objektivní metody se tedy snaží napodobit zrakový systém člověka (HVS) a je snaha o to, aby výsledky objektivních metod v co největší možné míře korelovaly s výsledky subjektivních testů. Dále se obě metody dají rozdělit na hodnocení obrazu s referenčním obrazem (FR-IQA), hodnocení obrazu bez referenčního obrazu (NR-IQA) a jejich kombinací (RR-IQA). Jelikož při focení snímků v aplikaci není možné rozhodnout, který obraz je referenční a který poškozený, jsem nucen použít jednu z objektivních metod hodnocení obrazu bez referenčního obrazu (NR-IQA). V následujících dvou kapitolách budou shrnuty popisy a výpočty některých subjektivní a objektivních metod pro vyhodnocení kvality obrazu. [3].

3 Subjektivní hodnocení kvality obrazu

Subjektivní (od slova subjekt), neboli osobní hodnocení kvality obrazu je založeno na hodnocení několika osob (pozorovatelů). Subjektivní hodnocení obrazu tedy znamená, že hodnocení kvality obrazu se vztahuje k danému jedinci, čili jak daná osoba obraz vnímá, nebo jaký z něj má pocit. Výsledky jsou tedy zpravidla u každé osoby rozdílné.

Norma ITU-R BT.500-13 z roku 2012 [4], jenž vydává mezinárodní telekomunikační unie, předepisuje doporučení pro subjektivní hodnocení kvality obrazu a videa. Norma je velmi podrobná a definuje několik možných měření. Základním předpokladem je výběr skupiny alespoň 10 lidí (pozorovatelů), kteří prošli testem zraku. V ideálním případě je skupina složena z lidí s různým věkem, s různou profesí apod. Následně je skupině v laboratoři na přesném zobrazovači promítnuta série několika fotek. Pozorovatelé v průběhu promítání hodnotí snímky na předem dané stupnici, výsledkem je pak průměrné hodnocení. Test by se měl několikrát opakovat, kvůli statistickým chybám. Norma taktéž udává, jak dlouho by měl test trvat, jaké by mělo být osvětlení v laboratoři, z jaké vzdálenosti by se měl pozorovatel dívat apod. [3].

Subjektivní metody lze rozdělit na 2 základní skupiny: DS Metody (pozorovatelé hodnotí kvalitu dvou podobných obrazů) a SS metody (pozorovatelé hodnotí pouze kvalitu samostatného obrazu bez reference).

Obrázek 2: Laboratoř pro subjektivní hodnocení kvality obrazu
(Převzato z [5])

3.1 Metody DS (Double Stimulus)

Metody, při kterých jsou pozorovatelům promítány 2 různé snímky, jeden referenční a druhý zkreslený testovacím systémem (se zhoršenou kvalitou).

3.1.1 DSIS (Double Stimulus Impairment Scale)

Pozorovatelé mají předem daný časový úsek, kdy mohou pro danou dvojici snímků hlasovat. Hlasování probíhá na stupnici od 1 do 5, kdy 1 znamená velmi nepříjemné zkreslení (tedy nejhorší možné zkreslení) a 5 znamená neznatelné (pozorovatel nezaznamenal žádné zkreslení). Nakonec

je výsledek zprůměrován ze všech možných měření. Metoda je vhodná pro zjištění práhu, kdy už je zkreslení člověkem znatelné. [1, 4].

3.1.2 DSCQS (Double Stimulus Continual Quality Scale)

Metoda je velmi podobná metodě DSIS. Avšak u DSCQS pozorovatel neví který obraz je referenční a který zkreslený. Pozorovateli se promítnou opět 2 různé snímky, ale tentokrát hodnotí oba snímky vůči sobě na stupnici od 0 do 5, kdy 0 znamená špatná kvalita a 5 výborná kvalita snímku. [1, 4].

3.2 Metody SS (Single-Stimulus)

Metody, u kterých se testuje pouze snímek bez potřeby reference. Pozorovatelům jsou tedy promítány snímky nezávisle na ostatních snímcích.

3.2.1 SSM (Single Stimulus Method)

Pozorovateli je promítnut snímek a opět hodnotí kvalitu dle svého uvážení. Celý test se může až 3× opakovat, aby se zamezilo statistickým chybám. K hodnocení kvality obrazu existují 2 různé metodiky [1, 4].:

- Nepřímá - Hodnotí se na stupnici od -3 (nejhorší kvalita) do +3 (nejlepší kvalita)
- Nekategorická - Hodnotí se na plynulé stupnici bez číslování nebo na stupnici od 0 do 100

3.2.2 SSCQE (Single Stimulus Continuous Quality Evaluation)

Tato metoda slouží především pro hodnocení kvality videa (pro zjištění kvality komprese MPEG apod.), ale dá se použít i pro hodnocení kvality obrazu. Opět je to metoda, kde se hodnotí scéna (snímky) bez reference. Pro metodu je doporučeno alespoň 15 pozorovatelů. Pozorovatelům je přehrávána sekvence a ti průběžně hodnotí kvalitu na stupnici od 0 do 100 (kde 0 je nejhorší kvalita a 100 nejlepší kvalita) na HW nebo SW posuvníku. Systém z tohoto posuvníku odečítá každé 0,5 sekundy hodnocení pozorovatele a tento test trvá většinou 10 až 20 minut. [1, 4].

3.3 Zhodnocení subjektivního testování

Subjektivní testování má výhody, ovšem ale i nevýhody. Výhody spočívají v tom, že kvalitu obrazu hodnotí člověk, výsledky jsou tedy přesné s HVS. Na základě těchto metod lze tedy určit hodnotu MOS (Mean Opinion Score), což je průměr hodnocení všech pozorovatelů a tato hodnota se porovnává s objektivními metodami, které se snaží v maximální možné míře korelovat s těmito hodnotami. Takto můžeme tedy získat velice přesný HVS model. Ovšem ze subjektivního testování vyvstávají i velké nevýhody jako: velká časová a finanční náročnost, dodržení přesných norem při testování v laboratořích, ale především nemožnost pružného nasazení do testovacích systémů.

4 Objektivní hodnocení kvality obrazu

Objektivní (od slova objekt), tedy neosobní, nebo předmětné hodnocení kvality obrazu. Od subjektivního hodnocení se liší tím, že objektivní hodnocení je nestranné (není závislé na člověku, či na skupině lidí). Výsledek u této metody hodnocení je tedy vždy stejný. Tyto metody se tedy snaží nahradit metody subjektivní a snaží se eliminovat nevýhody těchto metod. Přináší ovšem další nevýhody jako právě ono nestranné hodnocení a v současnosti nedokonalé algoritmy, které by se vyrovnaly subjektivním testům. [6].

Objektivní metody lze podobně jako u subjektivních metod rozdělit na 3 základní skupiny: FR metody (Full Reference - k výpočtu je použit referenční i testovaný obraz), NR metody (No Reference - k výpočtu je použit pouze jeden testovaný obraz bez reference), RR metody (Reduced Reference - k výpočtu je použita pouze část informací z referenčního obrazu i testovaný obraz). V této práci se budeme bavit pouze o metodách FR a NR.

4.1 Metody FR (Full Reference)

Tyto metody lze srovnat s DS metodami u subjektivního testování. Metody s plnou referencí jsou nejjednodušší, nejstarší a nejrozvinutější metody pro objektivní hodnocení kvality obrazu. Ke svému hodnocení potřebují jak referenční obraz, tak obraz testovaný (zkreslený obraz). I přestože v této práci nejsou tyto algoritmy použity, je nutné je alespoň z části zmínit.

4.1.1 MSE (Mean Squared Error)

Střední kvadratická chyba je nejjednodušší pixelově orientovaná metoda. I přes svou nedokonalost a fakt, že nerespektuje HVS je stále nejpoužívanější metrikou pro hodnocení kvality obrazu v reálných aplikacích. MSE je definována jako střední hodnota druhých mocnin rozdílů dvou jasových hodnot pixelů. Pro 2D obraz je MSE definován takto:

$$MSE = \frac{1}{MN} \sum_{x=1}^{M} \sum_{y=1}^{N} ||I(x,y) - \hat{I}(x,y)||^2 \tag{1}$$

Kde *I(x,y)* a $\hat{I}$*(x,y)* představují 2 různé obrazy o velikosti M × N (jeden referenční, druhý testovaný), resp. jasové hodnoty pixelů na souřadnicích *(x,y)*. Jelikož jeden obraz je referenční a druhý zkreslený, jejich rozdílem pak zjistíme hodnotu chybového signálu. MSE lze tedy považovat za měřítko kvality obrazového signálu. Tento přístup lze použít pouze u obrazu v odstínech šedi (hodnoty 0 - 255). Pokud bychom chtěli vypočíst MSE i pro barevný obraz (RGB), je nutno vzorec upravit přidáním sumy sčítající chyby barevných složek obrazu, kde k představuje právě tuto barevnou složku.

$$MSE = \frac{1}{3MN} \sum_{k=1}^{3} \sum_{x=1}^{M} \sum_{y=1}^{N} ||I(x,y) - \hat{I}(x,y)||^2 \tag{2}$$

Jelikož srovnáváme každý pixel referenčního obrazu s korespondujícím pixelem obrazu zkresleného je tato metoda výpočetně nenáročná a velmi rychlá. Tato metoda nám tedy vypočte obecnou informaci jak moc je obraz zkreslen, avšak ne kde a jakým způsobem, protože je srovnáván pixel s pixelem nezávisle na ostatních hodnotách pixelů. I přesto, že jsou i jiné metriky založeny na MSE, není MSE ideální metodou pro výpočet objektivního hodnocení kvality obrazu. [6, 7].

4.1.2 PSNR (Peak Signal-to-Noise Ratio)

Jedna z metrik odvozená z MSE. Maximální poměr signálu k šumu udává poměr mezi maximální možnou energií signálu a energií šumu. Spolu s MSE je to nejčastější metoda pro určování kvality obrazu bez referenčního obrazu. Tato metody byla vytvořena z důvodů, srovnání obrazů s rozdílnými dynamickými rozsahy, vyjadřuje tedy věrnost obrazu oproti referenčnímu. Pro 2D obraz je PSNR definován takto:

$$PSNR = 10log_{10}\left(\frac{MAX^2}{MSE}\right) \tag{3}$$

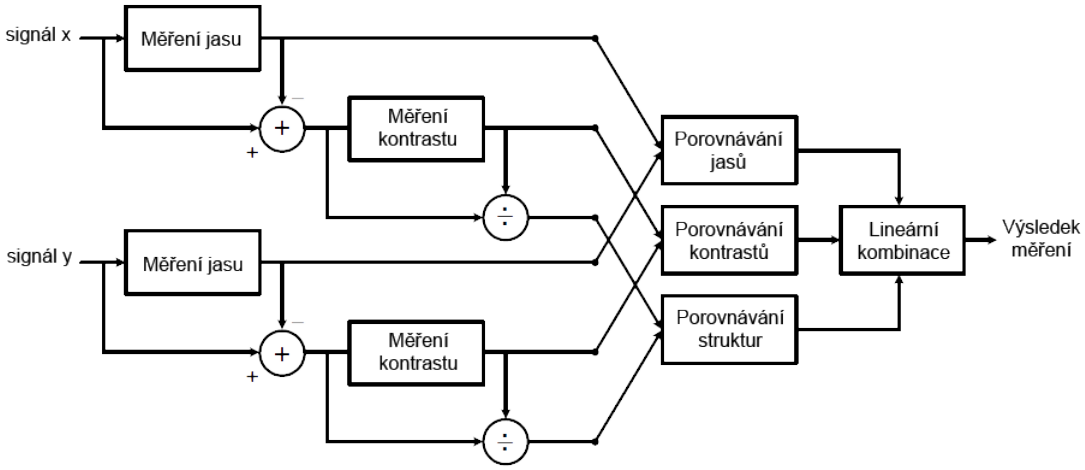
Kde MAX představuje maximální možnou hodnotu, které může nabývat pixel v obraze (např. u obrazu v odstínech šedi to je 255). PSNR je vyjádřeno v jednotkách *dB*. Čím větší hodnota, tím kvalitnější obraz. Kvalitní obraz nabývá hodnot cca 50 dB, komprimovaný mezi 30 a 40 dB a pro totožné obrazy je PSNR nedefinováno. I přesto, že PSNR taktéž není založen na HVS modelu, je někdy považován za dobrého ukazatele hodnocení kvality obrazu, jelikož dosahuje dobrých výsledků s porovnáním se subjektivními metodami (korelace s MOS někde mezi 60% - 80%). Proto může být PSNR použit jako objektivní metoda pro hodnocení kvality obrazu. [6, 7].

4.1.3 SSIM (Structural Similarity Index)

SSIM je jedna z nejlepších metrik pro hodnocení kvality obrazu, jelikož je již postavena aby zohledňovala lidské vnímání obrazu (HVS). Je tedy vylepšením metod MSE a PSNR, které tuto vlastnost nerespektovaly. Hodnocení z algoritmu nabývá hodnot 0 až 1, kde 0 znamená nulový vztah k referenčnímu obrazu a 1, že jsou obrazy totožné. Metoda je založena na měření podobnosti dvou obrazů podle tří složek, a to: podobnost jasu, podobnost kontrastu a strukturální podobnost, viz. obrázek 3. SSIM můžeme vypočítat podle vztahu:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma} \tag{4}$$

Kde člen *l(x,y)* porovnává jas signálů, *c(x,y)* porovnává kontrast signálů a *s(x,y)* porovnává strukturální korelaci. Koeficienty $\alpha > 0$, $\beta > 0$ a $\gamma > 0$ následně určují váhu důležitosti těchto tří členů. [1, 7].



Obrázek 3: Schéma systému měření SSIM

Toto byly 3 nejdůležitější metody pro hodnocení kvality obrazu. Metod existuje daleko více, ale v této práci se o nich nemá smysl nadále zmiňovat. Další metody lze nalézt např. v publikacích: [6, 7].

4.2 Metody NR (No Reference)

Tyto metody lze srovnat s SS metodami u subjektivního testování. Metody pro měření kvality obrazu bez reference nepotřebují žádný referenční obraz, tudíž pro měření stačí mít pouze testovaný obraz. Problémem těchto metod je zjistit co je v obraze zkreslení (šum, rozmazání, artefakty v obraze apod.) a co je „původní“ nezkreslená informace (nezkreslený obsah obrazu). Proto je snaha najít ideální model HVS na predikci těchto zkreslení, což je v dnešní době velký problém. Jelikož neexistuje univerzální algoritmus, zavádí se algoritmy, které se snaží najít pouze určitý typ zkreslení nebo jejich kombinaci (šum, rozmazání, artefakty v obraze apod.) a toto zkreslení vyhodnotit. [1].

V podstatě existují 3 základní přístupy k problému NR-IQA [8]:

- **Nalezení konkrétního rušení**

  Tyto algoritmy hledají v obraze určité rušení. Hledají již výše zmiňovaný šum, rozostření apod. Tento přístup je nejvíce využíván u hodnocení kvality MPEG videa. Algoritmy hledají blokové artefakty ve scéně pomocí DCT (diskrétní kosinové transformace), jelikož se toto rušení u MPEG kodeku předpokládá.

- **Přístup extrakce vlastností a učení se z něj**

  V tomto přístupu jsou extrahovány informace z obrazu a algoritmy se učí rozhodovat o tom, co je rušení, a co rušení není. V těchto algoritmech se využívají především neuronové sítě.

- **Přístup založený na NSS (Natural Scene Statistics)**

  Tento přístup spoléhá na to, jak se statistika obrazu změní, pokud jsou na něm použita některá rušení. Přístup je založen na předpokladu, že nezkreslené snímky zabírají subprostor celého prostoru všech možných obrazů. Algoritmus má pak snahu nalézt vzdálenost od zkresleného obrazu (obraz, který obsahuje rušení) do subprostoru nezkreslených snímků. Tento přístup je ze všech přístupů nejsložitější, avšak vypadá zatím nejslibněji, jelikož velmi dobře koreluje s výsledky HVS. [12].

Těmito přístupy se zabývá Laboratoř pro obrazové a video inženýrství (LIVE), která sídlí na univerzitě v Austinu v Texasu[1]. Pro tuto diplomovou práci je tedy nasnadě použít jeden z těchto přístupu. Byl vybrán přístup založený na NSS, díky jeho dobrým výsledkům při měření. Abychom ale mohli určit, který algoritmus zvolit, je potřeba si několik těchto metod popsat a následně je otestovat na sekvenci obrázků, aby byla vybrána nejvíce efektivní metoda.

4.2.1 NIQE (Naturalness Image Quality Evaluator)

Nejjednodušší pixelově orientovaná metoda založena na NSS, která ke svému měření nepotřebuje absolutně žádné referenční obrazy, ani žádné informace o tom, jak by měl vypadat zkreslený obraz (metoda tedy nevyužívá žádné databáze obrazů). Metrika měří index pouze tak, že hledá odchylky od statistických zákonitostí pozorovaných v obraze. Výsledky ukazují, že NIQE je velmi rychlý algoritmus pro hodnocení kvality obrazu. [13].

4.2.2 BRISQUE (Blind/Referencless Image Spatial QUality Evaluator)

Opět metoda, pracující v prostorové doméně. Tato metoda nepočítá zkreslení obrazu jako: šum, rozmazání, blokové artefakty apod., ale namísto toho používá NSS model k výpočtu normalizovaných jasových koeficientů v určitém místě obrazu, aby následně vypočetla ztrátu na „přirozenosti“ celého obrazu. BRISQUE je mnohem statisticky lepší než FR-IQA metody jako PSNR a SSIM. Je výpočetně rychlá, tudíž je možno ji nasadit v real-time aplikacích. [14].

4.2.3 DIIVINE (Distortion Identification Image Verity INtegrity Eval.)

Celým jménem: *Distortion Identification-based Image Verity and INtegrity Evaluation*. Využívá dvou přístupů ke hledání rušení. Především využívá přístupu extrakce vlastností z obrazu, přičemž využívá určité vlastnosti z NSS. Princip metriky spočívá ve dvou fázích a to, že se nejprve identifikuje zkreslení postihující obraz a následně se provede zkreslení specifické pro hodnocení kvality. Tato metoda již pracuje ve spektrální doméně, proto se předpokládá menší výkonnost algoritmu. [15].

[1] `http://live.ece.utexas.edu/`

Předešlé metody byly popsána jen velmi stručně, protože nejsou pro tuto práci až tak podstatné. Jsou ve své podstatě velice složité, a princip fungování je velmi obsáhlý. Bližší informace lze nalézt např. v publikacích [13, 14, 15].

4.2.4 BIQI (Blind Image Quality Index)

Poslední z popisovaných metod pro hodnocení kvality obrazu bez reference (NR-IQA). Taktéž metoda založena na principu NSS. Tato metoda byla pro práci vybrána pro své dobré výsledky s porovnáním se subjektivním hodnocením, které jsem prováděl a které je popsáno v následující kapitole.

BIQI pracuje na jednoduchém principu. Nejprve je algoritmu předán testovaný obraz, který nemá žádnou referenci a je u něj předpokládán určitý druh rušení. Tento obraz je následně převeden do stupňů šedi (256 odstínů), jelikož barevný model není pro nalezení rušení potřeba a také zvyšuje paměťovou a časovou náročnost algoritmu. BIQI se v obraze snaží najít tyto druhy rušení:

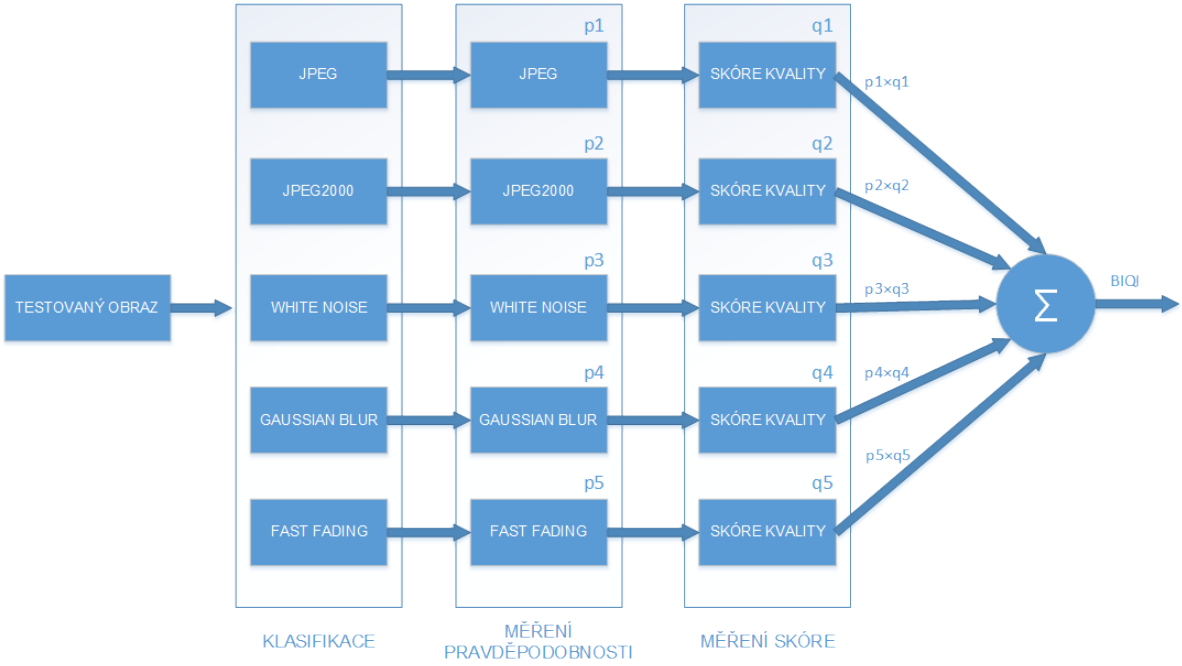
- **JPEG** - rušení obrazu, které vzniklo kompresí JPEG. Toto rušení je typické pro bloky 8×8 pixelů, jelikož při JPEG kompresi je použito diskrétní kosinové transformace (DCT).
- **JPEG2000 (JP2K)** - rušení obrazu, které vzniklo kompresí JPEG2000. Toto rušení je typické pro artefakty kolem ostrých hran, jelikož JPEG2000 komprese také pracuje ve spektrální doméně (jako JPEG komprese), akorát s tím rozdílem, že používá diskrétní vlnkovou transformaci (DWT).
- **White Noise (WN)** - tzv. bílý šum. Je to náhodný signál s konstantní hustotou výkonového spektra v obraze. Toto rušení může vzniknout při focení snímku.
- **Gaussian Blur (GB)** - neboli Gaussovské rozostření, nebo také „rozmázání“. Taktéž vzniká při vyfocení snímku kamerou a lze jej poznat tak, že je část obrazu nebo celý obraz „rozmazán“.
- **Fast Fading (FF)** - v doslovném překladu „rychlé blednutí nebo rychlý únik“. Na obraze lze toto rušení poznat prudkou změnou jasu nebo kontrastu mezi různými částmi obrazu.

K určení těchto rušení je použito datasetů z LIVE IQA[1], což je databáze snímků s předem přidaným rušením (JPEG, JP2K, WN, GB, FF), které následně byly subjektivně zhodnoceny, jak moc je snímek zkreslen (porušen). Pro vytvoření DIS (Distorted Image Statistics) modelů, které by obsahovaly informace o těchto rušeních, byly tyto snímky z databáze převedeny diskrétní vlnkovou transformací (DWT) do spektrálních koeficientů, který byly následně parametrizovány zobecněnou Gaussovou distribucí (GGD) [8].

[1] Ke stažení na: `http://live.ece.utexas.edu/research/quality/subjective.htm`

Pokud tedy budeme srovnávat obrazové informace testovaného obrazu s těmito DIS modely, můžeme následně získat informaci o tom, jaké typy rušení a v jaké míře obraz obsahuje. Toto poskytuje přenositelné řešení, jelikož jsou modely každého rušení uloženy v souborech, které tyto informace obsahují.

Na obrázku 4 lze vidět zjednodušený proces postupu při počítání indexu (hodnoty kvality) metody BIQI.



Obrázek 4: Blokové schéma algoritmu BIQI

Celý algoritmus probíhá ve dvou krocích. První krok spočívá v klasifikaci rušení na základě toho, jak se NSS změní po aplikování určitého druhu rušení na obraz. Provádí se tedy klasifikace rušení u obrazu a měření pravděpodobnosti výskytu tohoto rušení v obraze. K získání těchto informací je použito Support Vector Machines (SVM), což jsou metody, které rozdělují a porovnávají trénovací data (DIS modely) s informacemi v obraze. Tyto pravděpodobnosti ke každému rušení (JPEG, JP2K, WN, GB, FF) jsou tedy získány porovnáním dat z testovaného obrazu vůči DIS modelu, který obsahuje informace jaká je NSS při určitém druhu rušení. Pravděpodobnost se tedy vyhodnocuje pro každý typ rušení a na obr.4 je znázorněna jako p_i, $\{i=1,...,5\}$, kde i je typ rušení.

Po zjištění těchto pravděpodobností následuje samotné měření kvality obrazu. Měření je realizováno opět pomocí SVM (kromě JPEG komprese), kdy jsou obrazové informace srovnávány s DIS modely již určitého druhu rušení. Tímto získáme hodnotu vypovídající o míře daného rušení. Opět se počítá pro každé rušení zvlášť a na obr.4 je znázorněna jako q_i, $\{i=1,...,5\}$, kde i je typ rušení.

Nakonec jsou pravděpodobnosti typu rušení vynásobeny s korespondujícími naměřenými hodnotami hodnocení kvality a nakonec jsou všechny tyto hodnoty sečteny. Zjednodušený vzorec pro výpočet je tedy následující:

$$BIQI = \sum_{i=1}^{5}(p_i \cdot q_i) \tag{5}$$

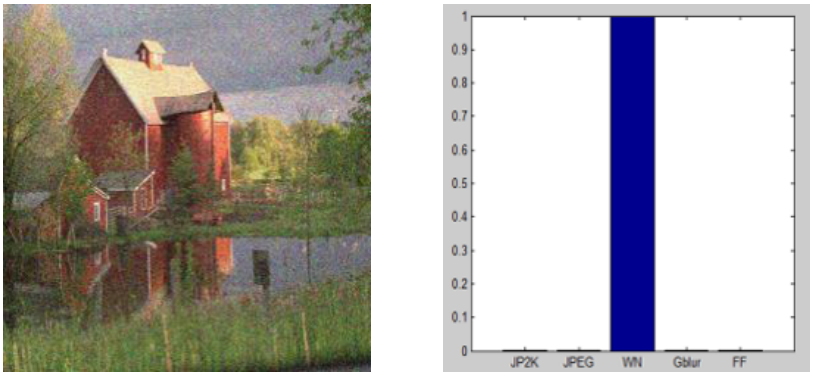
Již bylo zmíněno, že k určení kvality JPEG komprese není použito SVM. Pro vyjádření hodnoty kvality tedy nebylo zapotřebí žádného DIS modelu. K tomuto hodnocení kvality byl použit samostatný algoritmus, jelikož přístup s SVM nevykazoval příliš dobré výsledky v porovnání s HVS. Tento algoritmus na rozdíl od SVM přístupu, pracuje v prostorové doméně. V principu využívá toho, že se při kompresi obraz rozdělí na bloky 8×8 pixelů, nad kterými je provedena diskrétní kosinová transformace (DCT) a tyto bloky jsou následně kvantovány. Právě tímto rozdělením obrazu na nezávislé bloky vznikají v testovaném obraze především blokové artefakty, nebo také „ringing“ efekty, které v obraze zapříčiní artefakty kolem ostrých hran v podobě rozmazání. Algoritmus počítá 3 hodnoty: průměrnou změnu bloku oproti ostatním blokům, aktivitu obrazového signálu a hodnotu prudkých změn v obraze (zero-crossing). Všechny tyto hodnoty jsou počítány jak horizontálně, tak i vertikálně. Nakonec jsou všechny hodnoty zkombinovány pomocí modelu, který byl navrhnut pomocí subjektivních testování. Více informací o algoritmu lze nalézt v publikaci [10].

Pro úplné pochopení fungování algoritmu je potřeba zmínit ještě pár informací. Před samotnou klasifikaci obrazu se barevný obraz převede do stupňů šedi a poté do spektrální domény pomocí diskrétní vlnkové transformace (DWT) s použitím Daubechies 9/7 vlnky. DWT je použito, jelikož umožňuje získat časově-frekvenční popis obrazového signálu, a vlnka byla vybrána pro své dobré výsledky v kompresi a analýze obrazu [11]. Tato transformace je provedena ve třech stupních. Následně se tyto koeficienty musí parametrizovat pomocí nějaké spojité distribuční funkce. Jeden z jednoduchých modelů pro tyto koeficienty je zobecněná Gaussova distribuce (GGD), jedna z parametrických spojitých distribučních funkcí. Tato Gaussova křivka je vlastně funkce o třech parametrech: střední hodnoty μ, směrodatné odchylky σ a gamma parametru γ, které nám stačí pro klasifikaci typu rušení v obraze. Pro hlubší pochopení této problematiky jsou k dispozici publikace: [8, 9].

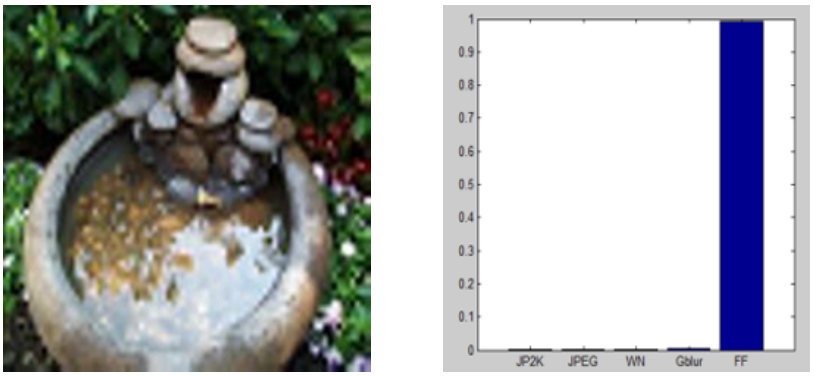
4.2.4.1 Výsledky BIQI

Výzkumníci z již zmiňované univerzity samozřejmě prováděli testy, jak si jejich algoritmus stojí v porovnáním se subjektivními testy. Toto srovnání je zde shrnuto, nicméně lze jej nalézt v [9]. Při provádění zhruba 1000 testů na testovacích vzorcích z LIVE IQA databáze vyšla korelace se subjektivními testy (tedy s HVS) 81.5161%. I přes takto vysoko korelaci jsem prováděl osobně subjektivní testy na skupině lidí, jelikož jsem se chtěl opravdu přesvědčit o těchto hodnotách a chtěl jsem je srovnat s již výše popsanými metodami, které jsou založeny na NSS.

Na následujících obrázcích je vždy vyobrazen testovací obrázek (vlevo), a k němu klasifikace rušení, tedy zařazení obrazu do určité třídy rušení s naměřenou pravděpodobností výskytu tohoto rušení v obraze za použití SVM.



Obrázek 5: Obrázek s rušením (WN) a jeho pravděpodobnost výskytu
(Převzato z [8])



Obrázek 6: Obrázek s rušením (FF) a jeho pravděpodobnost výskytu
(Převzato z [8])

V obou případech je tedy správně detekováno rušení v podobě *White Noise* a *Fast Fading*.

4.3 Zhodnocení objektivního testování

Objektivní testování tedy poskytuje zřejmé výhody. Především lze tyto metody nasadit do testovacích systémů a je jisté, že na rozdíl od subjektivního testování budou vždy poskytovat stejné výsledky. Vše ovšem závisí hlavně na výběru metriky. Jak již bylo řečeno, existují metriky, které vůbec nekorespondují s HVS, metriky, které řeší pouze určitá rušení v obraze a metriky, které velmi dobře korelují s HVS, ale jsou náročné na implementaci, nebo jsou velmi výpočetně náročné. Proto je velmi dobré věnovat značnou část času tomuto výběru.

5 Testování vybraných NR-IQA algoritmů

Pro výběr jednoho z NR-IQA algoritmů do této práce jsem nejprve musel provést srovnání těchto algoritmů se subjektivními testy. Pro toto testování bylo vybráno 30 snímků. Všechny tyto snímky byly vyfoceny mobilním telefonem *LG Nexus 5*, jelikož je dáno, že snímky budou v aplikaci pro Android vyfoceny právě fotoaparátem na nějakém mobilním zařízení. Tyto snímky jsem vybral, abych v co největší míře pokryl celou stupnici hodnocení, tzn., že jsem vybíral z mého hlediska jak „pěkné“ snímky, tak snímky „nepěkné“ (rozmazané, jednobarevné, přesvětlené, apod.). Poté jsem pro každý tento obrázek provedl objektivní měření (BIQI, NIQE, BRISQUE a DIIVINE), a každé toto měření poskytlo objektivní ukazatel kvality obrázku, tedy index kvality obrazu daného algoritmu.

Abych mohl zjistit, který z těchto algoritmů poskytuje nejlepší výsledky, musel jsem provést reálné subjektivní testy. Tyto testy byly prováděny na skupině 8 lidí a pro hodnocení byla vybrána nekategorická SSM metoda, kdy každý pozorovatel vybíral hodnotu ze stupnice 0 - 100. Tento test byl následně prováděn ještě jednou, aby bylo zamezeno statistickým chybám (tzn., že pozorovatel není schopen pokaždé určit stejnou hodnotu téhož snímku). Abychom určili hodnotu MOS (Mean Opinion Score), nejprve se zprůměrovala hodnocení pozorovatele k danému obrázku a poté i hodnocení všech pozorovatelů ke stejnému obrázku.

Ovšem získané hodnoty MOS i indexy kvality z objektivního měření jsou samostatně vůči sobě nic nevypovídající, jelikož každý člověk má svou subjektivní stupnici, s jejíž pomocí hodnotí. Tzn., že nemůžeme srovnávat hodnoty MOS přímo s indexy kvality z objektivního měření. Proto musíme tyto data podrobit korelační analýze. Tím zjistíme, zda hodnoty z objektivního měření závisí, nebo jsou závislé na hodnotách měření ze subjektivních testů (zjišťujeme tedy vzájemný vztah dvou proměnných). Pro všechna srovnávání (MOS hodnot s určitým objektivním měřením) byl vybrán Pearsonův korelační koeficient (označujeme r). Jeho výpočet je následující:

$$r = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}} \tag{6}$$

Kde i představuje číslo obrázku, čili x_i je hodnota MOS z množiny MOS hodnot a y_i je hodnota indexu kvality daného algoritmu z množiny těchto hodnot. $\overline{x}$ a $\overline{y}$ potom udávají průměr z těchto množin. Hodnota korelačního koeficientu nám udává, do jaké míry jsou na sobě hodnoty závislé. Hodnoty blížící se -1 značí zcela nepřímou závislost (antikorelaci) a hodnoty blížící se 1 značí přímou závislost (korelaci), ke které se chceme co nejvíce přiblížit.

Byly tedy srovnávány všechny 4 zmíněné algoritmy pro měření kvality obrazu, resp. jejich výsledné hodnoty se subjektivními hodnotami MOS. Tímto jsme získali 4 korelační koeficienty. Koeficient, který se nejvíce blížil číslu 1, vypovídá o tom, že nejlépe koresponduje s výsledky subjektivních testů (tedy se samotným HVS). Na následující straně jsou v tabulce 1 hodnoty MOS a vypočtené objektivní indexy kvality ke každému obrázku. V tabulce 2 jsou pak korelační koeficienty pro každý otestovaný algoritmus.

Poznámka 1 Všechny obrázky byly testovány v rozlišení 1280 × 720 a ve formátu *jpg* jsou k dispozici v příloze A.

Poznámka 2 Všechna subjektivní i objektivní měření jsou k dispozici ve formátu *xlsx* taktéž v příloze A.

Tabulka 1: Výsledky subjektivních a objektivních měření

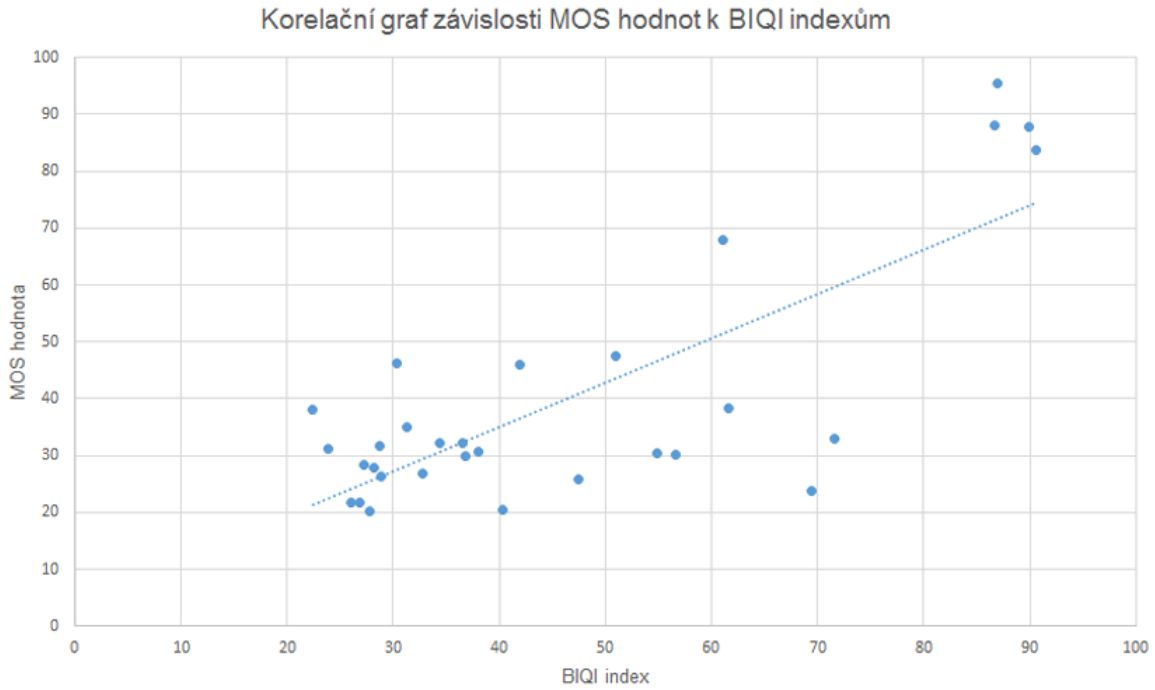
	MOS	**BIQI**	**NIQE**	**BRISQUE**	**DIIVINE**
Obrázek 1	26,0	21,6407	3,5676	33,6789	23,1956
Obrázek 2	28,25	27,8835	4,218	29,0628	29,2703
Obrázek 3	90,5625	83,7287	6,6606	38,9277	50,7338
Obrázek 4	30,3125	46,1652	5,1147	38,2106	36,4637
Obrázek 5	61,125	67,9153	6,8871	50,9744	39,9482
Obrázek 6	28,75	31,6074	4,655	33,3754	35,4379
Obrázek 7	47,4375	25,9423	5,9805	48,8413	46,3921
Obrázek 8	71,625	33,0818	6,4619	35,3447	53,2376
Obrázek 9	38,0	30,6605	5,0903	39,6104	48,61
Obrázek 10	54,9375	30,453	4,2077	33,224	33,5944
Obrázek 11	23,875	31,1213	4,0198	30,1113	34,5436
Obrázek 12	56,6875	30,11	8,9779	47,2829	54,524
Obrázek 13	50,9375	47,5285	7,3538	55,569	56,2921
Obrázek 14	27,25	28,2841	4,2497	31,0707	31,9013
Obrázek 15	40,3125	20,5395	3,2295	26,182	22,2193
Obrázek 16	36,875	29,7718	4,2036	29,001	40,7534
Obrázek 17	89,875	87,8923	7,0075	31,9768	28,7213
Obrázek 18	42,0	45,9381	5,384	42,8633	45,0186
Obrázek 19	36,5625	32,2288	3,8223	25,0399	25,5269
Obrázek 20	69,5	23,6536	6,0522	46,0204	33,5956
Obrázek 21	31,3125	34,9637	4,2749	35,0595	33,9931
Obrázek 22	28,8125	26,4278	3,6034	28,3123	26,3475
Obrázek 23	34,4375	32,0991	4,5753	39,5962	33,4562
Obrázek 24	86,75	88,1077	9,4892	41,8367	30,1207
Obrázek 25	86,9375	95,3378	8,1026	45,4629	28,1983
Obrázek 26	61,5625	38,387	4,7368	33,8663	35,1004
Obrázek 27	26,8125	21,6158	3,0664	23,7053	17,794
Obrázek 28	27,8125	20,2942	3,4043	22,7395	20,3715
Obrázek 29	32,75	26,7219	4,9638	42,5027	38,6832
Obrázek 30	22,4375	38,1679	4,6002	39,6077	35,3908

Tabulka 2: Korelační koeficienty objektivních testů ve srovnání s MOS

	BIQI	**NIQE**	**BRISQUE**	**DIIVINE**
r	0,7774	0,7797	0,4115	0,2845

Z tabulky 2 je patrné, že algoritmus BIQI má korelační koeficient 0,7774. To tedy znamená, že algoritmus BIQI ze 77,7% odpovídá vnímání kvality obrazu dle lidského zrakového systému, alespoň dle mého testování. Tímto jsem si tedy alespoň částečně potvrdil měření vycházející z [9], kde tato hodnota činila 0,8151. Pro přesnější výsledky bych musel patrně provést více takovýchto testů (vyšší počet pozorovatelů i vyšší počet testovaných obrazů). Nicméně pro srovnání s ostatními NR-IQA algoritmy tato měření bohatě postačují.

Na obrázku 7 lze vidět korelační graf mezi hodnotami MOS, získanými ze subjektivních měření a indexy kvality, získaných z algoritmu BIQI pro všech 30 testovaných obrazů. V grafu lze vidět přímku, která odpovídá hodnotě korelačnímu koeficientu 0,7774. Z hodnot z grafu lze také vidět korelaci, a to takovou, že pozorovatelé obvykle určovali hodnotu menší, než index kvality z objektivního měření u vyšších hodnot, naproti tomu určovali hodnotu větší, než BIQI index u nižších hodnot.



Obrázek 7: Korelační graf k hodnotám MOS a k BIQI indexům

Algoritmy BRISQUE a DIIVINE vykazovaly nejmenší korelaci s HVS. Jsou tedy velmi nepřesné a výstupy někdy až náhodné. Pro tuto diplomovou práci se tedy nehodí. Naopak BIQI a NIQE svou přesností překvapily a rozhodoval jsem se tedy mezi těmito dvěma algoritmy. Nakonec jsem si pro výběr „nejpěknější“ fotky ze série pořízených vybral algoritmus BIQI, který je sice nepatrně výpočetně složitější než zmiňovaný NIQE, ale naproti NIQE hledá BIQI v obraze více druhu rušení, u kterých je možnost nastavení vah.

6 Návrh aplikace

Prvotním a klíčovým bodem při tvorbě aplikace je jistě návrh. V této kapitole budou nejprve popsány obecné principy fungování obou aplikací (klient i server) a poté následuje část, ve které je rozebrán návrh klientské aplikace a posléze je uveden i návrh serverové komponenty.

6.1 Obecný popis

Prvním úkolem bylo vymyslet jméno aplikace, které zní: *GPS Photo Logger*. Název je tedy odvozen ze slov, která popisují tuto aplikaci. Následně je uveden a krátce shrnut obecný princip fungování celého systému (klientské i serverové části) a architektura tohoto systému.

6.1.1 Aplikace pro Android

Aplikace, která by měla sloužit jako klientská část celého systému. Po jejím nainstalování do mobilního zařízení, by měl mít uživatel možnost registrace do celého systému, tedy mohl by používat jak tuto aplikaci, tak i serverovou část. Po registraci by se uživatel mohl přihlásit do aplikace, přičemž by proběhlo ověření vůči serverové části, resp. vůči databázi, která je na serveru. Po přihlášení se předpokládá zobrazení přehledného menu, kde bude na výběr z několika možností jako: *spuštění logování*, *zobrazení webového rozhraní*, *zobrazení galerie* a *nastavení určitých funkcí aplikace*. Stěžejní část funkce aplikace se tedy nachází v možnosti logování. Při výběru této možnosti se zobrazí prostředí s mapovým podkladem, na kterém bude zaznamenána současná lokace zařízení a další informace (tedy podobně jako u aplikací, které slouží jako GPS navigace). V tomto prostředí bude mít uživatel 2 možnosti jak spustit automatické focení:

- **Spuštění snímání zatřesením**

  Snímání by se mělo spustit po zatřesení mobilního telefonu. Aplikace vyfotí daný počet snímků, z těchto snímků následně rozhodne, který snímek je nejlepší (algoritmem BIQI) a pošle ho na server spolu s informacemi, kde byl tento snímek pořízen, kdy byl pořízen apod. Hodnoty jako počet fotek a síla zatřesení by měly mít možnost změny v nastavení aplikace. Tato metoda je vhodná, pokud chce uživatel vyfotit zájmový bod náhle, bez předchozího přidání bodu na serverové straně.

- **Spuštění snímání podle aktuální polohy**

  Snímání by mělo být spuštěno, když se zařízení přiblíží poloze, resp. polohám, které si uživatel předem nastavil ve webovém rozhraní na serverové části. Z toho tedy vyplývá, že by se ze serveru měla stahovat data těchto lokací, ještě před započetím logování, tedy snímání fotek v závislosti na poloze. Při přiblížení k danému zájmovému bodu (k dané poloze) se opět spustí vyhodnocování fotek a nejlepší snímek se zašle spolu s informacemi na server. Pro použití této metody je vhodné zařízení umístit za čelní sklo do auta (popř. na kolo, motocykl apod.), spustit logování a projíždět danými body.

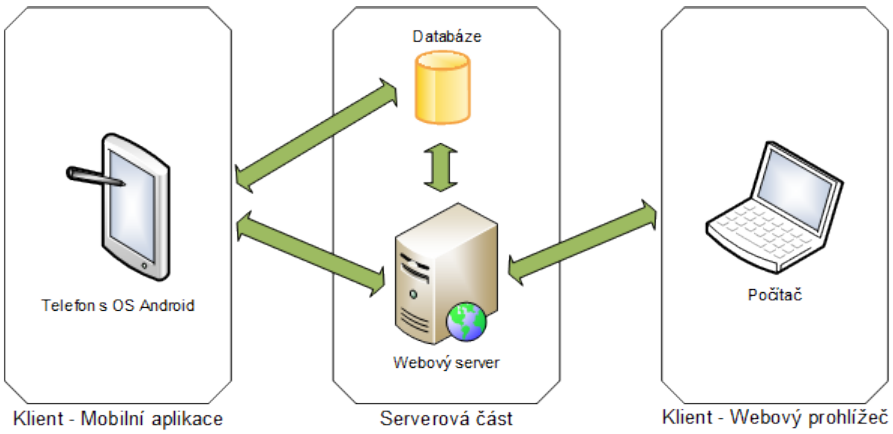
6.1.2 Webová aplikace

Aplikace, která by měla sloužit jako serverová část vytvářené aplikace. Úkolem je vytvořit rozhraní, neboli aplikaci pro webový server, se kterým by komunikovala klientská aplikace pro Android. Tento server by měl splňovat následující kritéria:

- Na serveru by měla být databáze, která by poskytovala možnost uložení samotných uživatelů (jejich jména spolu s hesly), aby bylo možné ověření těchto uživatelů v obou částech aplikace. Také by se zde měly ukládat informace o zájmových bodech, které si uživatel navolil.
- V aplikaci by měl mít uživatel možnost navolit si své lokace zájmových bodů přímo na mapovém podkladu. Tyto informace by se ukládaly do databáze.
- Uživatel by měl mít také možnost prohlížení pořízených záznamů vyfocených fotek, které byly poslány z klientské aplikace pro Android.
- Komunikace klienta se serverem by měla být zabezpečena pomocí HTTPS protokolu. Tedy přenášená data (fotky a informace zájmového bodu) by měla být šifrována pomocí SSL nebo TLS.

6.1.3 Architektura systému

Na obrázku 8 je vidět zjednodušená architektura celého systému. Klientská aplikace pro Android komunikuje s databází za účelem ověřování identity pro přihlašování a také přímo z databáze stahuje uživatelovy uložené zájmové body. Rovněž komunikuje přímo s webovým serverem pomocí HTTPS protokolu kde pomocí dotazovací metody POST zasílá nejlepší vybranou fotografii spolu s informacemi ve formátu JSON. Klient - webový prohlížeč tedy komunikuje s webovým serverem pouze za účelem zobrazení obsahu (taktéž šifrovaně pomocí HTTPS).



Obrázek 8: Architektura systému

6.2 Klientská část - aplikace pro Android

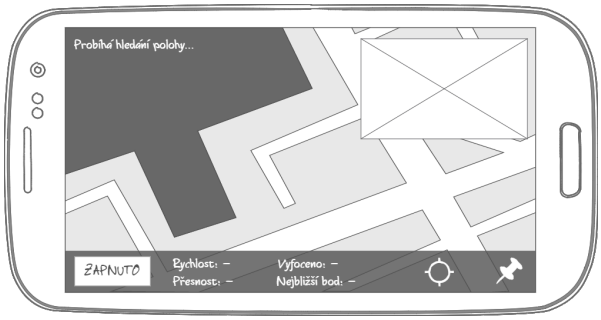
V následující sekci je popsán návrh aplikace pro Android. Prvně je popsáno uživatelské rozhraní, tedy skica neboli náčrtky aplikace. Poté její fungování, interakce s uživatelem a nakonec požadavky aplikace.

6.2.1 Návrh uživatelského rozhraní

Nejprve bylo zapotřebí navrhnout uživatelské rozhraní aplikace, bylo tedy nutné načrtnout „mock-up“ prototypy (tedy skicy jak by měla aplikace přibližně vypadat). Na samém začátku jsem musel zvolit, jak bude vypadat hlavní aktivita aplikace (úvodní obrazovka aplikace pro Android). Z důvodů, že aplikace bude fungovat pouze po registraci a přihlášení uživatele, musí tedy logicky úvodní obrazovka vypadat jako přihlašovací dialog, kde by uživatel zadal údaje a přihlásil se. Taky by zde měl možnost přejít na obrazovku registrace. Po úspěšném přihlášení by se mělo zobrazit hlavní menu. Všechny tyto návrhy obrazovek jsou vyobrazeny na následujícím obrázku 9.

Obrázek 9: Návrh obrazovek aplikace (Přihlášení, Registrace, Hlavní menu)

Z obrázku lze vidět, že je kladen velký důraz na design, přívětivost a jednoduchost aplikace. Byla snaha vše navrhovat ve stylu *material designu*, což je „designový jazyk“ navrhnut firmou *Google*. Je tedy používaná stejná sada ikon, barvy jsou jemné, jasné apod. V menu budou tedy 4 možnosti výběru. Na obrázku 10 lze vidět hlavní obrazovku pro logování. Dále je zde na výběr tlačítko pro zájmové body, kde se pouze spustí prohlížeč se stránkou webového serveru. Ikona galerie zobrazí galerii vyfocených obrázku a při kliknutí na ikonu nastavení uživatel přejde na obrazovku, kde si bude moct nastavit chování aplikace.

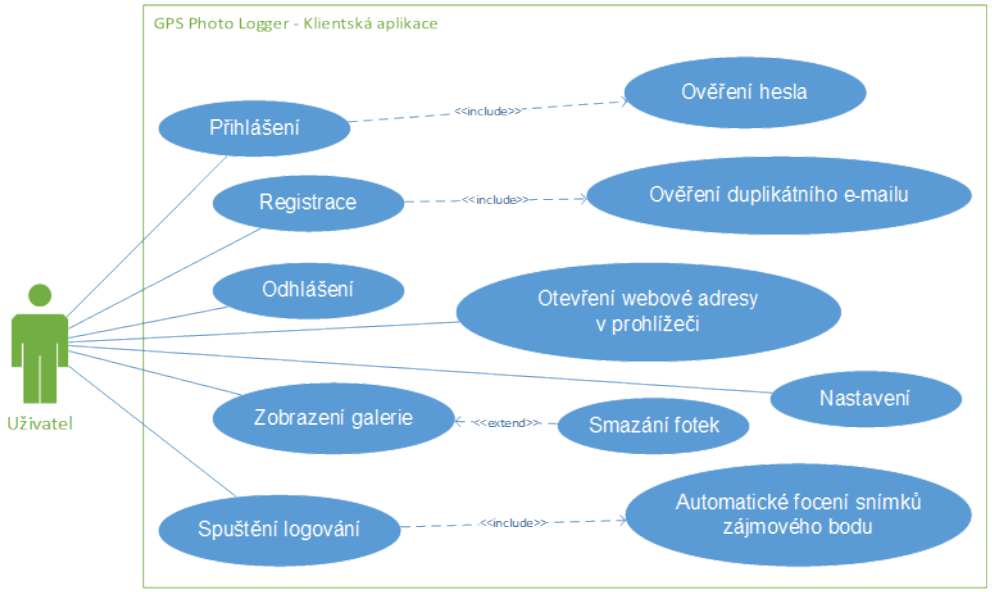


Obrázek 10: Návrh obrazovky pro logování

Na obrázku 10 lze vidět návrh hlavní obrazovky pro logování. Jako pozadí je zvolen mapový podklad *Google Maps*, který se dynamicky mění, v závislosti na poloze zařízení. V horním pravém rohu bude náhled scény, kterou aktuálně bude snímat kamera zařízení. V dolní části zleva se postupně nachází: tlačítko pro zapnutí automatického pořizování fotek v závislosti na poloze zařízení, následně informace o rychlosti, přesnosti polohy, počtu vyfocených snímků a vzdálenost k nejbližšímu bodu. Vpravo dole jsou potom ikony na zjištění aktuální polohy a zobrazení zájmových bodu, které se budou fotit.

6.2.2 Funkční specifikace

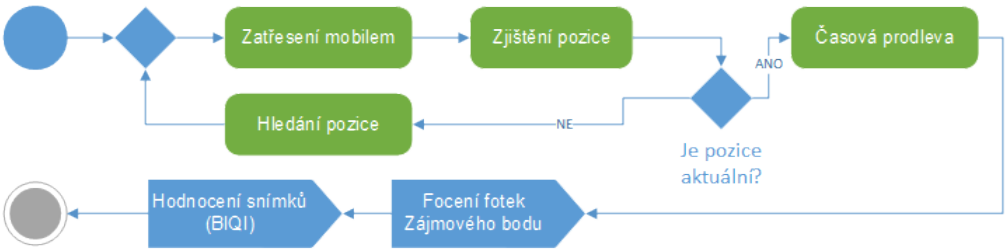
Po načrtnutí prototypů uživatelského rozhraní je zapotřebí vědět jaké má aplikace funkce, jak se chová a co může uživatel v aplikaci dělat. K tomu slouží např. diagram užití (use case diagram), který je vyobrazen na následujícím obrázku 11.



Obrázek 11: Diagram užití klientské aplikace pro Android

Hlavním účelem celé aplikace je ale spuštění funkce (zatřesením zařízení, nebo dle polohy), která by automaticky začala pořizovat fotky a po pořízení těchto fotek by již zmíněný algoritmus BIQI vybral z objektivního hlediska nejlepší fotku. Ke zjednodušenému popisu těchto funkcí jsou vyobrazeny diagramy aktivit na obrázcích 12 a 13.

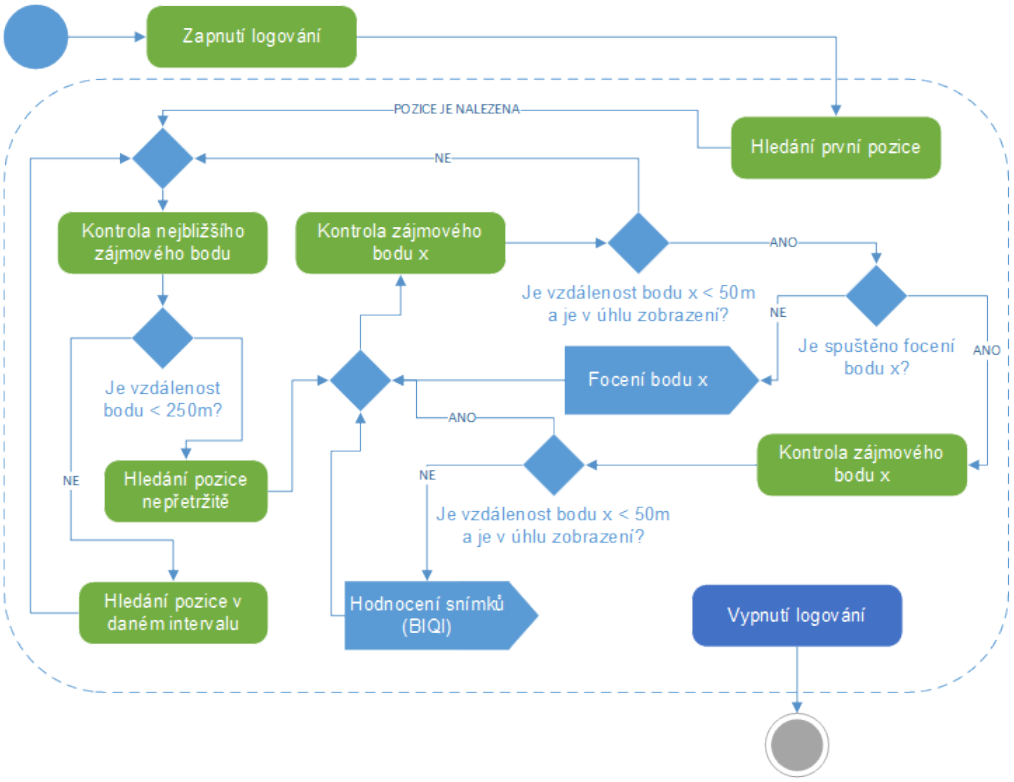
U prvního případu, tedy pořizování fotek na základě zatřesení zařízení je scénář velmi jednoduchý. Pro spuštění celého procesu je zapotřebí, aby uživatel provedl s mobilním telefonem jakýkoli pohyb (sílu akcelerace bude možno předem zvolit v nastavení aplikace). Následuje vyhledání polohy. Pokud je poloha aktuální, vyčká aplikace 2 sekundy (jelikož by se při pohybu mobilem mohly vyfotit rozmazané snímky) a automaticky vyfotí předem nastavený počet snímků. Po vyfocení těchto snímků se spustí algoritmus BIQI pro nalezení nejlepší fotky. Ovšem pokud není poloha aktuální, musí se nalézt, což může někdy trvat delší dobu. Proto po nalezení polohy musí uživatel opět zatřást telefonem.



Obrázek 12: Diagram aktivit pořizování fotek zatřesením zařízení

Ovšem největší důraz v aplikaci byl kladen na funkcionalitu automatického pořizování snímků na základě polohy. Zde je již scénář poněkud složitější. Není asi třeba zmiňovat, že pro fungování této funkce je potřeba mít zapnutou GPS v telefonu. Další podmínkou pro focení zájmových bodů je nutnost mít předem nějaké tyto body přidané z rozhraní webové aplikace. Po zapnutí logování musí telefon vyhledat první pozici. Dále se v daném intervalu hledá poloha zařízení a kontroluje se vzdálenost od nejbližšího zájmového bodu. Pokud je tato vzdálenost větší než 250 metrů, stále se vyhledává poloha zařízení pouze v daném intervalu. Ovšem pokud je zařízení blíže, než 250 metrů k tomuto bodu, zpřesní se vyhledávání polohy, a to tak, že jsou polohy aktualizovány jak jen to je možné. Tento způsob hledání polohy je navrhnut z důvodu, že není třeba neustále zjišťovat polohu zařízení, když nejsou nablízku zájmové body a především z důvodu úspory energie. Následně je kontrolována vzdálenost všech zájmových bodů. Pokud je tato vzdálenost u určitého bodu menší, než 50 metrů (a zároveň je v úhlu zobrazení k tomuto bodu), začne aplikace automaticky pořizovat snímky pro tento bod, dokud se zařízení od zájmového bodu nevzdálí na vzdálenost větší než 50 metrů (nebo bude mimo úhel zobrazení). V aplikaci bude možnost tento úhel nastavit, popř. úplně vypnout toto rozhodování. Po vyfocení těchto snímků se tedy opět může spustit algoritmus BIQI na vyhodnocení nejlepšího obrázku z vyfocených.

V diagramech aktivit není zakresleno posílání nejlepšího vybraného snímku na server, jelikož je tato funkce řešena samostatně. A to způsobem, že bude navrhnuta služba, která bude kontrolovat nově zapsaný soubor v zařízení a bude také zjišťovat aktuální konektivitu zařízení. Podle těchto údajů pak aplikace pošle snímek spolu s informacemi na server.



Obrázek 13: Diagram aktivit pro pořizování fotek dle polohy zařízení

6.2.3 Technická specifikace

Pro korektní fungování aplikace jsou na mobilní telefon kladeny tyto požadavky:

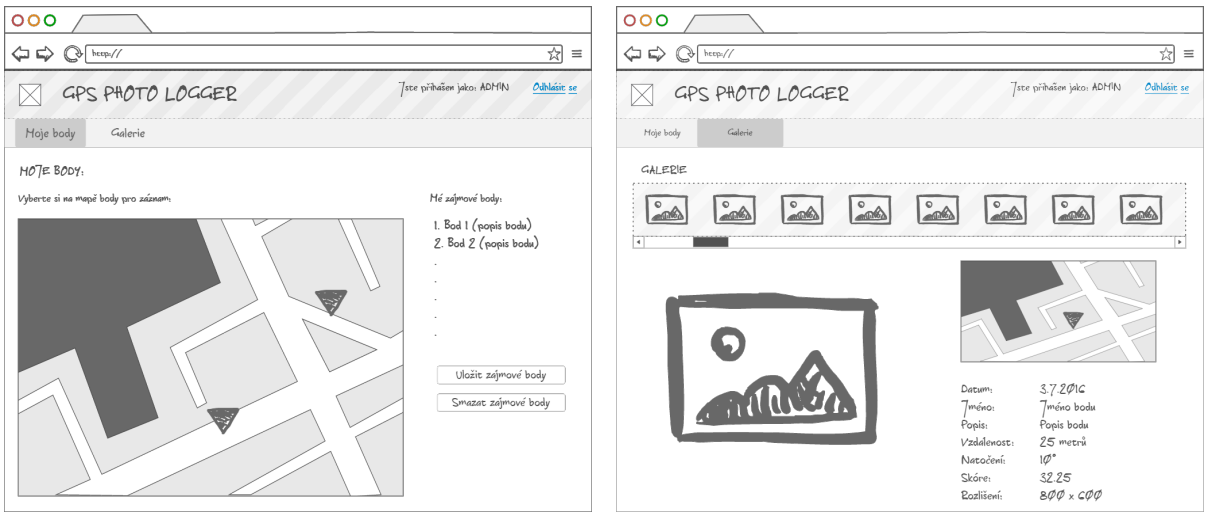
- Minimální verze OS Android - 4.0.3
- Mobilní telefon musí disponovat kamerou a GPS modulem
- Telefon také musí disponovat Akcelerometrem pro zatřesení mobilem, a také Magnetometrem pro určení natočení zařízení vůči severu
- V úložišti zařízení musí být dostatek místa na ukládání pořízených fotek zájmových bodů a souborů, které budou obsahovat informace o tomto bodu

6.3 Serverová část - webová aplikace

V následující podkapitole je popsán návrh serverové komponenty. V první řadě je zde opět vyobrazen a popsán náčrt uživatelského rozhraní, dále funkce a chování této aplikace, a nakonec návrh samotné databáze.

6.3.1 Návrh uživatelského rozhraní

Opět se předpokládá, že úvodní webová stránka, která by se uživateli zobrazila jako první, bude vypadat jako přihlašovací dialog. Pokud je tedy uživatel registrován, může se přihlásit do webového rozhraní. Po přihlášení se bude moci pohybovat pouze po dvou jednoduchých stránkách, a to po stránce pro zadávání zájmových bodů a stránce pro prohlížení pořízených fotografií zaslaných klientskou aplikací. Na následujícím obrázku 14 je vyobrazeno, jak by tyto stránky mohly vypadat.



Obrázek 14: Návrh webových stránek

- **Stránka pro přidání zájmových bodů (nalevo)**

  Jednoduchá stránka, kde si uživatel bude moct zadat zájmové body. Po kliknutí na mapový podklad se zobrazí dialog, který nabídne zadání jména zájmového bodu a jeho popisu. Po zapsání těchto údajů se objeví nový bod na mapě a také jméno spolu s popisem vedle mapy. Až si bude uživatel jist, že zadal všechny body, které chtěl přidat, klikne na tlačítko *Uložit zájmové body* a všechny zájmové body se uloží do databáze. Samozřejmostí je také možnost mazání zájmových bodů.

- **Stránka pro prohlížení galerie (napravo)**

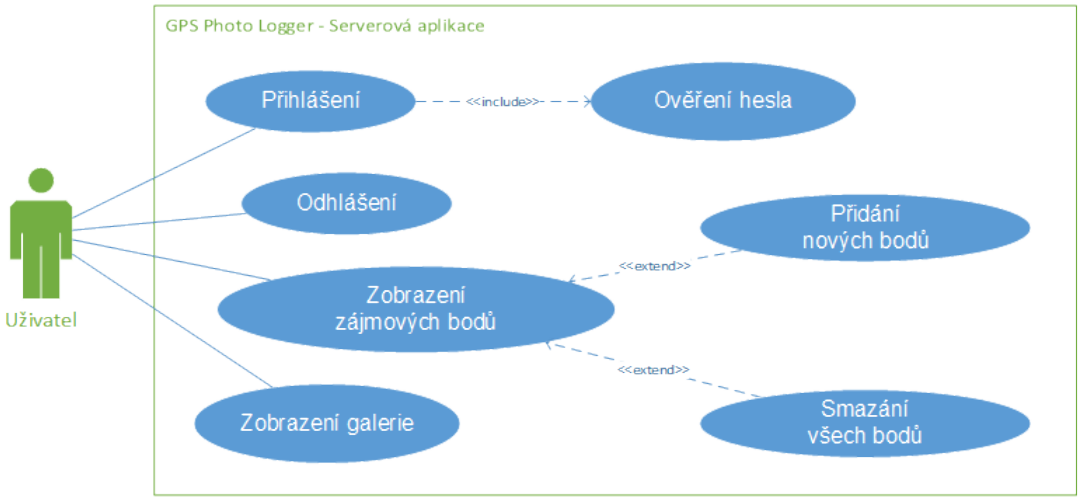
  Stránka, která bude umožňovat prohlížení zaslaných „nejlepších“ fotek z klientské aplikace, spolu se zobrazením informací o této fotce. V horní části je seznam fotografií (miniatur těchto fotek), které byly zaslány klientskou aplikací a jsou uloženy na serveru. Po kliknutí na určitou miniaturu se v dolní části zobrazí fotografie spolu s informacemi zájmového bodu, kde byla tato fotografie pořízena. Informace budou obsahovat:

  – Datum, kdy byla fotografie pořízena
  – Název a popis zájmového bodu
  – Vzdálenost a natočení zařízení ve chvíli vyfocení fotografie od zájmového bodu
  – Skóre fotografie (neboli index kvality vyhodnocen algoritmem BIQI)
  – Rozlišení fotografie

  Spolu se zobrazením informací o fotografii (resp. zájmovém bodu) se také na mapovém podkladu zobrazí lokace v závislosti na tom, jak byl snímek pořízen. Pokud byl snímek pořízen zatřesením zařízení, zobrazí se na mapě lokace v podobě šipky ve směru v jakém bylo zařízení v době pořizování fotografie. Pokud byl snímek pořízen v závislosti na detekci polohy, zobrazí se na mapě lokace zájmového bodu a také šipka udávající směr, v jakém byla fotografie pořízena.

6.3.2 Funkční specifikace

V předchozí podkapitole již byly stručně popsány funkce, kterými serverová aplikace disponuje. Na obrázku 15 jsou tedy tyto funkce přehledně zobrazeny v diagramu užití.



Obrázek 15: Diagram užití serverové aplikace

6.3.3 Databáze

Nedílnou součásti serverové komponenty je také samozřejmě databáze. V této podkapitole je tedy potřeba navrhnout samotnou databázi. Tedy tabulky (a vztahy mezi nimi), kde se budou ukládat uživatelova data.

Již bylo řečeno, že databáze, resp. SŘBD (Systém Řízení Báze Dat) bude oddělena od webového serveru. To nám může poskytovat určitou přenositelnost SŘBD. Tedy pokud by webové stránky byly na jiném serveru než SŘBD a zároveň by byl webový server nedostupný, pořád by byl uživatel schopen stáhnout zájmové body z databáze, která by běžela na samostatném serveru. Jelikož tato funkcionalita nebyla požadována v diplomové práci, je webový server a SŘBD umístěn na jednom serveru. Hlavním úkolem je ale návrh dvou jednoduchých databázových tabulek. Ten je vyobrazen na následujícím obrázku 16.

Obrázek 16: Návrh databázových tabulek

Nalevo je tedy tabulka pro uložení uživatele, kvůli jeho registraci a následnému ověření v aplikacích. Jako první je ID, které bude pro každého uživatele jedinečné, bude primárním klíčem a bude se automaticky inkrementovat. Následně je potřeba zaznamenat také jeho jméno a e-mailovou adresu, která bude taktéž jedinečná a bude se pomocí ní identifikovat v aplikacích. Nakonec je potřeba samozřejmě zaznamenávat heslo, aby se uživatel verifikoval vůči aplikaci. Toto heslo bude kvůli bezpečnosti v databázi uloženo jako hašovací funkce.

Napravo je návrh tabulky pro uložení zájmového bodu. Jako první je zapotřebí opět uchovávat ID, které bude pro každý zájmový bod jedinečné a bude primárním klíčem tabulky. ID se bude opět automaticky inkrementovat s každým dalším přidaným záznamem. Následně je třeba uchovat jméno a popis zájmového bodu. Toto jsou tedy údaje, které zadává uživatel po přidání zájmového bodu na mapě. Nakonec také musíme definovat souřadnice zájmového bodu a to pomocí zeměpisné šířky a zeměpisné délky tohoto bodu.

7 Implementace aplikace

Hlavním výstupem této diplomové práce bylo ovšem naimplementovat aplikaci, tedy její klientskou i serverovou stranu. Kapitola je opět rozdělena do dvou částí, kde jsou rozebrány a vysvětleny postupy implementace důležitých částí programu. Informace jsem čerpal z [16, 17, 18].

7.1 Klientská část - aplikace pro Android

Důležitým faktorem při implementaci aplikace je výběr správného vývojového prostředí, tzv. IDE, ve kterém se vyvíjí daný software. Pro vytváření této aplikace bylo použito IDE *Android Studio* (ve verzi 2.0), což je oficiální nástroj od společnosti *Google*, který slouží pro vytváření právě Android aplikací, založen na vývojovém prostředí známém jako *IntelliJ IDEA*.

Na začátku samotné implementace je třeba si říci, pro jakou verzi Android SDK budeme aplikaci vyvíjet. K datu, kdy byla psaná tato diplomová práce, bylo nejnovější SDK ve verzi 23 (Android 6.0). Je tedy zřejmé, že aplikace bude vyvíjena a kompilována pro tuto verzi. Nyní je nasnadě určit minimální verzi SDK. To bylo vybráno ve verzi 15 (Android 4.0.3). Znamená to tedy, že aplikaci si spustí uživatel, který vlastní mobilní telefon s Androidem 4.0.3 a výše. Důvod výběru této verze spočívá v tom, že v době psaní této práce je tato verze OS nainstalována přibližně na 97,3% všech zařízení s OS Android, což pro účely této diplomové práce plně postačuje. Aplikace a její zdrojové kódy jsou k dispozici v příloze A.

7.1.1 Oprávnění aplikace

Pro správný a bezproblémový chod aplikace je zapotřebí povolit určitá oprávnění při instalaci aplikace. Pokud má cílové zařízení verzi OS Android 6.0 a výše, povolují se jisté z těchto oprávnění až za běhu aplikace, jelikož právě od této verze přibyla nová funkcionalita s názvem „Runtime Permission“. Tato oprávnění se volají v aplikaci za použití třídy `ContextCompat` pomocí metody `checkSelfPermission(String permission)`, kde řetězec `permission` je dané oprávnění. Aplikace tedy využívá následující oprávnění:

- `android.permission.INTERNET` - Umožňuje aplikaci otevírat síťové sockety. Používá se samozřejmě při oboustranné komunikaci se serverem.
- `android.permission.ACCESS_NETWORK_STATE` - Oprávnění ke zjištění stavu sítě. Použití např. u rozhodování, zda se může fotografie spolu s informacemi poslat na server.
- `android.permission.CAMERA` - Oprávnění k použití integrované kamery zařízení.
- `android.permission.ACCESS_FINE_LOCATION` - Umožňuje aplikaci určit přesnou polohu zařízení. Využití zejména při logování.
- `android.permission.WAKE_LOCK` - Zabraňuje telefonu, aby přešel do režimu spánku. Využívá se, aby aplikace neusnula při zapnutém logování.

- `android.permission.WRITE_EXTERNAL_STORAGE` - Umožňuje zápis na externí uložiště. Použití při ukládání vyfocených fotografií a informací o zájmovém bodu.
- `android.permission.READ_EXTERNAL_STORAGE` - Umožňuje čtení z externího uložiště. Využití např. v galerii při načítání fotek a informací.

Všechna tato oprávnění jsou zapsána v *Android Manifestu*, což je aplikační manifest, který obsahuje informace o celé aplikaci. Mimo jiné např. deklaruje aktivity aplikace, Intent filtry, služby apod.

7.1.2 Struktura aplikace

Celá aplikace byla implementována s ohledem na přehlednost. Třídy aplikace jsou tedy uloženy v jednom hlavním balíčku, který obsahuje přehledně rozdělené další balíčky.

- `cz.luki.gpsphotologger` - hlavní balíček, kde jsou převážně aktivity aplikace, ale i třída uchovávající cesty složek pro ukládání fotografií a dalších souborů. V hlavním balíčku se nacházejí také další balíčky (prefix `cz.luki.gpsphotologger` byl vynechán):
  - `biqi` - Třídy, kterými je implementován zmiňovaný algoritmus BIQI. Tento balíček obsahuje další 2 balíčky:
    - `svm` - Třídy pro SVM, které využívají knihovnu *libsvm*
    - `wavelets` - Obsahuje pouze jednu třídu pro reprezentaci Daubechies 9/7 vlnky
  - `data` - Převážně třídy pro komunikaci se serverem (připojení k databázi, nahrávání souborů na server), ale také třída pro práci s lokální SQLite databází
  - `gallery` - Třídy pro uchovávání hodnot v galerii (fotografie, informace)
  - `photologger` - Nejdůležitější balíček, ve kterém jsou uloženy třídy, které řeší funkčnosti jako: vyhledávání pozice, focení fotografií, bufferování těchto fotografií, zjišťování stavu, zda je možno poslat fotografii na server apod.

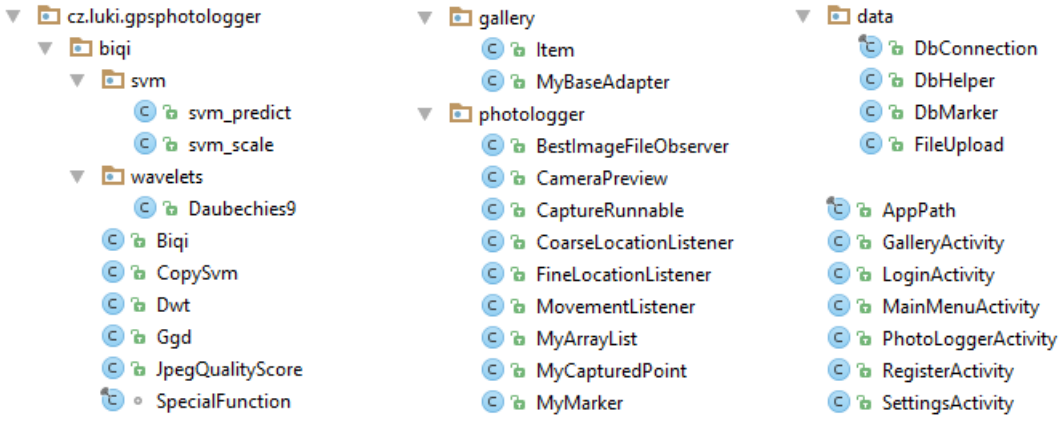
Reprezentaci těchto balíčku a tříd lze vidět na obrázku 17 na následující straně.

7.1.2.1 Použité knihovny

Při vytváření aplikace byly také použity 2 knihovny, a to:

- ***jtds-1.3.1*** - Knihovna pro připojení k Microsoft SQL databázi, a pro práci s ní [1]. Je volně ke stažení pod licencí *LGPL (GNU Lesser General Public License)*.
- ***libsvm-3.20*** - Knihovna pro funkci Support Vector Machines (SVM) [2]. Je volně ke stažení pod licencí *BSD (Berkeley Software Distribution)*.

[1] Ke stažení na: `https://sourceforge.net/projects/jtds/files/`
[2] Ke stažení na: `https://www.csie.ntu.edu.tw/~cjlin/libsvm/#download`



Obrázek 17: Reprezentace balíčků a tříd v klientské aplikaci

7.1.3 Přihlášení a registrace

Ihned po spuštění aplikace se zobrazí první aktivita, kterou reprezentuje třída `LoginActivity`. Tato jednoduchá třída obsahuje 2 widgety `EditText` pro zadání e-mailu a hesla. Pod nimi je tlačítko pro přihlášení, a pod ním je jednoduchý `TextView` pro přechod na obrazovku s registrací. Hlavní funkce aktivity spočívá v kliknutí na tlačítko „Přihlášení“. Po kliknutí na tlačítko se nejprve zkontrolují údaje, zda je vyplněn e-mail, zda má správný formát a jestli je vyplněno heslo. Pokud jsou všechny tyto podmínky splněny, následně se pomocí metody `saveToPrefs(Context, String, String)` uloží e-mail do *Shared Preferences* do vnitřní paměti zařízení, aby se tento e-mail automaticky předvyplnil při každém dalším přihlašování. Hned poté se zobrazí prvek `ProgressBar`, abychom byli informováni o stavu, že probíhá přihlašování. Nakonec se vytvoří nový objekt `UserLoginTask`, dědící z abstraktní třídy `AsyncTask`, který spustí asynchronní vlákno. Tuto část funkce lze vidět v následujícím výpisu.

```
if (cancel) {
   focusView.requestFocus();
} else {
   saveToPrefs(LoginActivity.this, PREFS_EMAIL, email);
   tv_caption.setText(getString(R.string.signing_in));
   showProgress(true);
   authTask = new UserLoginTask(email, password);
   authTask.execute((Void) null);
}
```

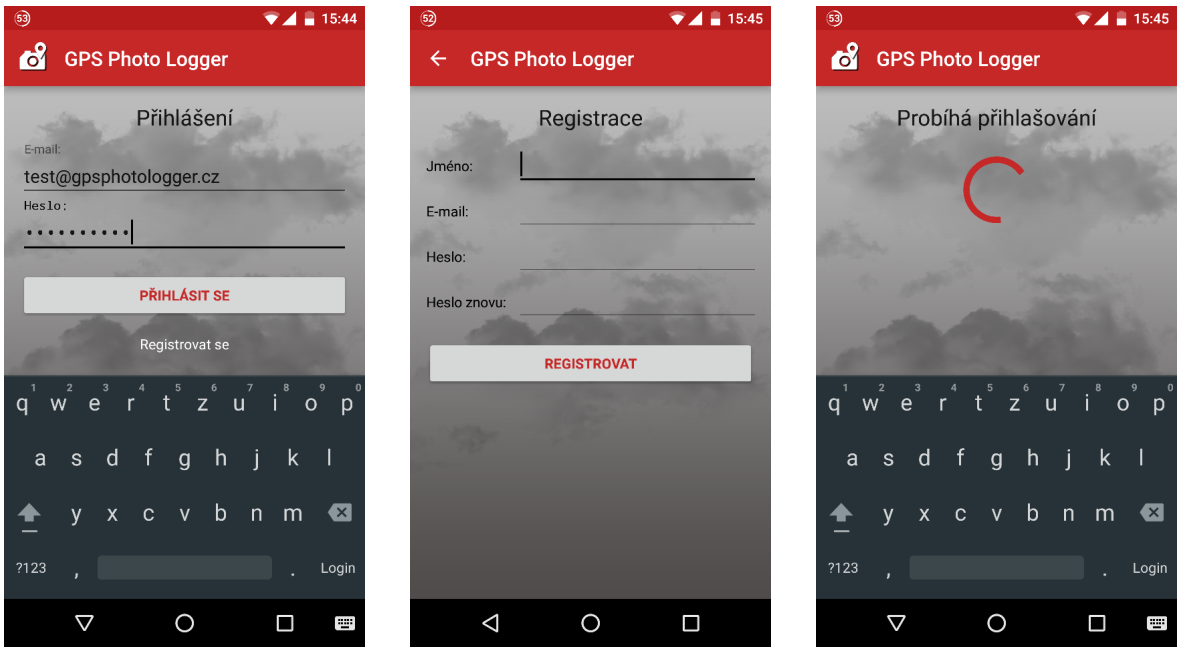
Výpis 1: Přihlašování v aplikaci

V tomto vlákně se aplikace připojuje k databázi na serveru a následně zjišťuje, zda jsou zadané údaje shodné s údaji v databázi.

Využití třídy `AsyncTask` je nutné, jelikož připojení k databázi může trvat delší dobu a v tomto časovém úseku by nemělo být blokováno hlavní vlákno aplikace. Pokud nejsou údaje správné, nebo se nelze připojit k databázi, zobrazí se varovná hláška za pomocí třídy `Toast`. Ovšem v opačném případě se vytvoří nový `Intent` (záměr) a s pomocí třídy `Intent` za využití metody `putExtra(String, String)` se přidají extra data v podobě ID, jména a e-mailu uživatele. Nakonec se tedy zobrazí aktivita `MainMenuActivity`.

Ovšem pokud doposud není uživatel registrován, nemůže se logicky do aplikace přihlásit. Kliknutím na prvek `TextView` „Registrovat se“ se zobrazí aktivita `RegisterActivity`, jejíž rodičovská aktivita je `LoginActivity`. Aktivita je vytvořena obdobně jako aktivita pro přihlašování s jedním zásadním rozdílem, že ve vlákně `AsyncTask` (konkrétně `RegistrationLoginTask`) se namísto zjišťování údajů, zjišťuje zda je v databázi uživatel se stejným e-mailem, či nikoli. Pokud je e-mail volný, vloží se do databáze nový uživatel, který je nyní registrován.

Na následujících obrázcích lze vlevo vidět obrazovku pro přihlášení uživatele, uprostřed obrazovku registrace a napravo je zobrazena obrazovka při procesu přihlašování.



Obrázek 18: Přihlašování a registrace v aplikaci pro Android

7.1.3.1 Účet Administrátora Pro účely testování je do aplikace zahrnut „účet Administrátora“. Pro přihlášení je potřeba zadat e-mail *admin* a heslo také *admin*. V aplikaci je tato funkčnost implementována tak, že se vůbec nespouští třída `AsyncTask`, tudíž se nekontrolují údaje v databázi. To nám poskytuje tu výhodu, že můžeme aplikaci využít i v případě nedostupnosti serveru. „Administrátor“ může také jako jediný v nastavení aplikace měnit IP adresu serveru. To je vhodné zejména při přesunutí webové aplikace na jiný server.

7.1.4 Hlavní menu

Aktivita, která reprezentuje hlavní menu, dostala název `MainMenuActivity`. Po spuštění této aktivity se převezmou extra data (ID, jméno, e-mail), která byla zaslána z předchozí aktivity `LoginActivity` a uloží se pro další zpracovávání. Obzvlášť důležité následné využití v aplikaci má ID, které se uloží do členské proměnné `userID` v právě instanciované statické třídě `UserID`. Přihlašování tedy funguje tak, že jakmile se jednou uživatel do aplikace přihlásí, bude přihlášen dokud se sám neodhlásí, nebo pokud nebude ukončena aktivita (proběhne metoda `onDestroy()`). V metodě `onBackPressed()`, tedy když uživatel stiskne tlačítko zpět, se samotná aktivita přesune pouze na pozadí pomocí metody `moveTaskToBack(true)`.

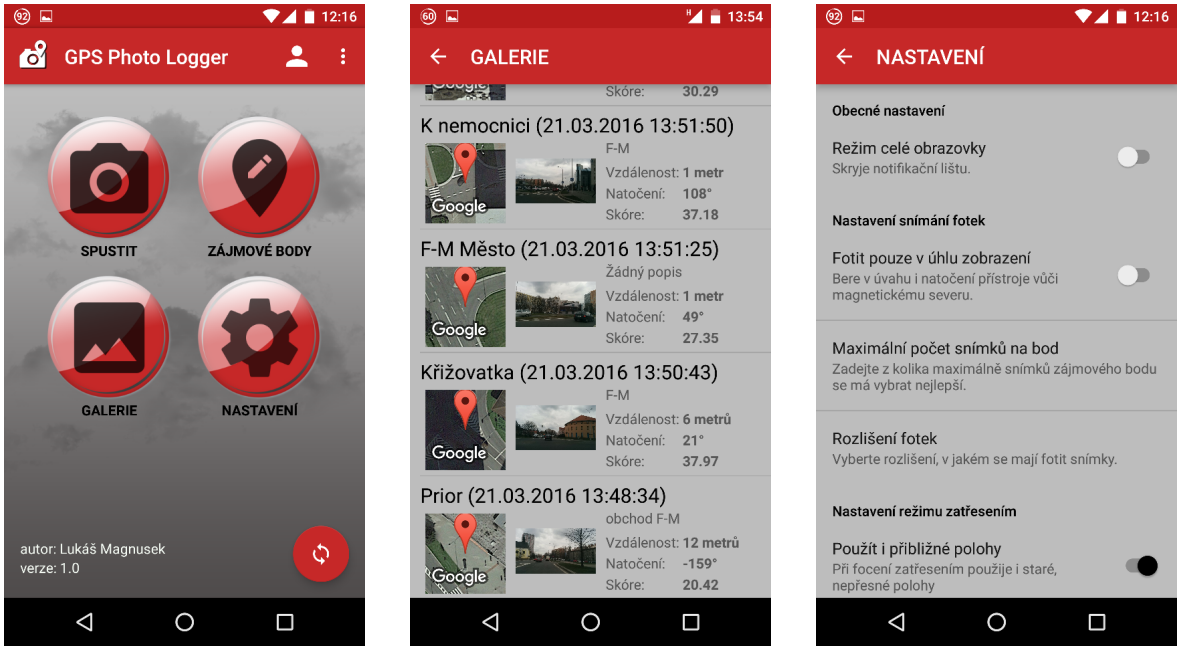
Ihned po uložení uživatelských údajů do proměnných se aplikace snaží stáhnout zájmové body ze serveru. Toto je řešeno opět pomocí třídy `AsyncTask` (konkrétně `DownloadDataTask`) podobně jako u přihlašování nebo registrace, s tím rozdílem, že se z databáze načtou pouze uživatelovy zájmové body. Při jejich čtení se vytváří pro každý bod nový objekt třídy `DbMarker`, který má stejné členy jako sloupce v dané databázové tabulce. Jedná se tedy o objektově relační mapování. Tyto objekty jsou následně uloženy do lokální SQLite databáze pro pozdější využití. Tato funkcionalita se tedy provádí v metodě `onCreate()`, avšak pouze při prvním vytvoření aktivity, kdy je předávaný parametr `savedInstanceState` null (žádný). Tímto se zamezí, aby se data stahovala např. při každé změně orientace obrazovky (jelikož `onCreate()` se spouští právě i po změně orientace). Nicméně celou tuto funkcionalitu stahování zájmových bodů řeší ikona v pravé dolní části aktivity, která je implementována za pomocí třídy `FloatingActionButton`.

Po provedení těchto částí kódu je ještě potřeba zkopírovat DIS modely pro potřeby SVM z aplikační složky *assets* do interní paměti zařízení. To je provedeno v metodě `copySvmFiles()`, avšak pokud má zařízení Android 6.0 a vyšší, předchází tomuto metoda `requestPermissions()`, která zajistí přidělení práv pro zápis na do úložiště zařízení. Rovnou se aplikace ptá i na přidělení práv pro užití kamery pro pozdější užití.

Z předchozí kapitoly víme, že v hlavním menu jsou k dispozici 4 funkce: *spuštění logování*, *otevření webového rozhraní*, *zobrazení galerie* a *nastavení*. Tyto funkce reprezentují 4 kulaté ikony v hlavním menu a jsou v aktivitě zobrazeny pomocí widgetu `ImageView`. Po kliknutí na vybranou ikonu se spustí příslušná obrazovka (aktivita). Tyto obrazovky budou následně popsány.

7.1.5 Nastavení

Obrazovka reprezentována třídou nesoucí jméno `SettingsActivity`. Třída je poděděna třídou `PreferenceActivity`, jelikož se v aktivitě využívají preference, tedy nastavení. V aktivitě je možnost nastavení těchto voleb: zda se použije režim celé obrazovky, zda bude aplikace pořizovat fotografie pouze v úhlu zobrazení, nebo jestli se budou u focení zatřesením používat i přibližné polohy. Pokud se jedná o nastavení hodnot, lze nastavit: maximální počet snímků na bod, rozlišení fotografií a sílu zatřesení mobilem. Dále lze nastavit také IP adresa serveru (může měnit pouze admin) a také lze stiskem na preferenci otestovat připojení k serveru.



Obrázek 19: Hlavní menu, galerie a nastavení v aplikaci pro Android

7.1.6 Galerie

Po kliknutí na ikonu *Galerie* se zobrazí galerie s vyfocenými fotografiemi zájmového bodu spolu s informacemi. I když tato funkcionalita v zadání diplomové práce nebyla požadována, usoudil jsem, že by byla velmi vhodná a pro uživatele zajímavá. Při vytváření aktivity `GalleryActivity` se nejprve načtou všechny soubory ve formátu *bin* ze složky *GPSPhotoLogger/bin/* (tuto cestu stejně jako ostatní lze nastavit ve třídě `AppPath`), což jsou vlastně serializované instance třídy `MyCapturedPoint`, které obsahují informace o zájmovém bodu. Po deserializaci souboru se načte i příslušná fotografie zájmového bodu a tato fotografie se s vybranými údaji z objektu `MyCapturedPoint` uloží do nově vytvořeného objektu `Item`. Tato třída `Item` představuje informace obsažené v jednom řádku galerie. Tyto objekty se následně uloží se do kolekce implementující rozhraní `List`, konkrétně `List<Item>` s názvem `items`. Po načtení všech tříd `Item` se musí kolekce seřadit podle data pořízení od nejnovější pomocí následující části kódu:

```
Collections.sort(items, new Comparator<Item>() {
   @Override
   public int compare(Item lhs, Item rhs) {
      return (rhs.getDate()).compareTo(lhs.getDate());
   }
});
```

Výpis 2: Řazení objektů galerie dle data

Tímto docílíme, že nejnovější vyfocené zájmové body budou zobrazeny v aktivitě nahoře. Celý návrh aktivity je tedy realizován za pomocí třídy `ListView`, kde jeden řádek představuje již zmiňovaná třída `Item`. Aby se ale data zobrazila, je využito třídy `MyBaseAdapter`, která je odvozena od třídy `BaseAdapter` a slouží k přiřazení údajů z objektů třídy Item do layoutu řádku prvku `ListView`. Aktivita `GalleryActivity` dále poskytuje následující interakce:

- Po kliknutí na mapu se otevře příslušná aplikace na zobrazení map (*Google Maps*).
- Po kliknutí na obrázek se otevře aplikace pro prohlížení souborů (pokud je v zařízení nějaká takováto aplikace k dispozici) a otevře složku, kde jsou uloženy všechny vyfocené snímky daného zájmového bodu. Toto se hodí zejména při testování, jelikož názvy souborů těchto fotek obsahují i skóre hodnocení algoritmem BIQI.
- Při dlouhém podržení (pomocí metody `onItemLongClick()`) na daný řádek galerie (prvek `Item`) se zobrazí dialog pro smazání fotky a všech příslušných souborů. Po výběru možnosti „Ano“ se tedy všechny tyto soubory smažou.

Na obrázku 19 lze uprostřed vidět obrazovku galerie.

7.1.7 Zobrazení webového serveru

Po kliknutí na ikonu *Zájmové body* se pouze otevře aplikace na prohlížení webových stránek v zařízení s danou adresou. Tato adresa se skládá z protokolu HTTPS, poté z IP adresy serveru a nakonec je přidán parametr *email* s e-mailem uživatele (tato část kódu je realizována pomocí třídy `Intent` a je zobrazena ve výpisu 3). Tento parametr na serveru zapříčiní, že se toto jméno automaticky předvyplní do pole E-mail a nastaví se *focus* na pole Heslo, aby ho uživatel mohl rovnou napsat.

```
Intent iLocation = new Intent(Intent.ACTION_VIEW);
iLocation.setData(Uri.parse("https://" + DbConnection.SERVER_IP + "/?email=" +
    userEmail));
startActivity(iLocation);
```

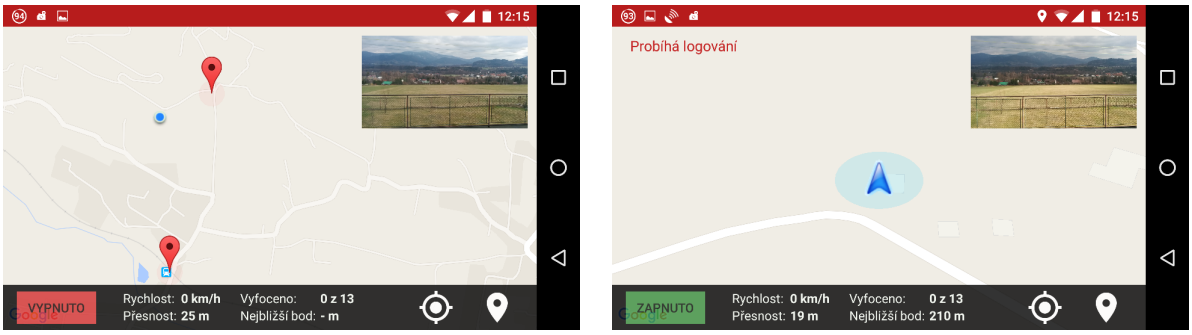
Výpis 3: Otevření stránek webového serveru

7.1.8 Pořizování fotek (Logování)

Stěžejním úkolem byla ovšem implementace funkce automatického pořizování fotek na základě polohy zařízení nebo zatřesením tohoto zařízení. Celá funkcionalita v aplikaci probíhá v aktivitě `PhotoLoggerActivity` a v přidružených třídách, které se nacházejí v balíčku `photologger`. Implementace těchto funkcí je poměrně rozsáhlá, proto budou postupně a stručně popsány implementace těchto tříd v následujících několika bodech.

7.1.8.1 PhotoLoggerActivity

Stručný popis a návrh hlavní aktivity `PhotoLoggerActivity` byl již popsán v předchozí kapitole, zde je pouze stručně shrnuta implementace této aktivity. Na obrázku 20 lze vidět jak tato aktivita vypadá, pokud je logování zájmových bodů vypnuto (nalevo), a když je zapnuto (napravo).



Obrázek 20: Aktivita logování v aplikaci pro Android

Aktivita nejprve zavolá metodu `onCreate()`, kde se inicializují různé proměnné, vytvářejí se nové instance tříd, načítá se nastavení ze *SharedPreferences*, je vytvořena instance SQLite databáze pro načtení zájmových bodů, inicializují se kontrolní prvky a listenery (posluchače událostí) aktivity a nastavují se pole `captureThreads`, `biqiThreads`, `capturedMarker`, `bestBitmaps`, které budou popsány později. Také se zde vypíší stažené zájmové body do `PopupMenu`, které se zobrazí po stisknutí ikony „lokace“ vpravo dole.

Poté je volána metoda `onStart()`, kde se vytvoří instance třídy `BestImageFileObserver`, která sleduje, zda aplikace může poslat fotografie na server. Ovšem o této třídě bude psáno později. Nyní je pomocí metody `getCameraInstance()` vytvořena instance třídy `Camera`, která reprezentuje přístup k integrované kameře zařízení. Pokud je připojení k zadní kameře zařízení úspěšné, vytvoří se objekt třídy `CameraPreview`, na kterém je ihned spuštěn náhled kamery za pomocí metody `startPreview()`. V metodě `onStart()` je také nastavováno rozlišení fotografií, ve kterém bude kamera fotit, ale především je nutné zmínit nastavení úhlu pro focení. Za pomocí metod `getParameters().getHorizontalViewAngle()` se na základě velikosti snímače a objektivu kamery zjistí, v jakém horizontálním úhlu zobrazení je kamera schopna snímat obraz. Tato hodnota se v jednotkách stupňů uloží do statické proměnné `ANGLE_OF_VIEW`, nicméně nás bude zajímat hodnota `halfAngleBearing`. Pokud je v nastavení zvoleno, aby se fotilo pouze v úhlu zobrazení, nastaví se tato hodnota na polovinu hodnoty `ANGLE_OF_VIEW`, jelikož při určování úhlu vůči zájmovému bodu získáváme absolutní hodnotu úhlu k tomuto bodu (nezáleží tedy, zda je bod nalevo, či napravo, zajímá nás pouze, zda je v úhlu zobrazení). Ovšem pokud je nastaveno, aby se nefotilo v úhlu zobrazení, nastaví se hodnota na 180°a kamera bude moct snímat nezávisle na úhlu vůči zájmovému bodu. Tato hodnota nám tedy pomáhá určit, zda je zájmový bod v úhlu zobrazení, či nikoli.

Jelikož aktivita implementuje rozhraní `OnMapReadyCallback`, volá se ihned po načtení mapového podkladu metoda `onMapReady()`. Opět se zde inicializují různé proměnné, pomocí metody `addMarkersToMap()` se na mapu přidají zájmové body implementované pomocí třídy `Marker` a přidají se různé listenery. Především je zde instanciována třída `LocationManager` pro zjišťování polohy zařízení a následně jsou zaregistrovány třídy `CoarseLocationListener` a `FineLocationListener`, které jsou popsány v dalších bodech. Především je ale důležité vědět, že je ihned zavolána metoda `getCoarseLocation()`, která se snaží najít poslední, nebo alespoň přibližnou pozici zařízení. Jakmile je tato pozice nalezena, může se dále s aktivitou pracovat.

Nyní máme 2 možnosti jak s aktivitou naložit. Můžeme zatřást telefonem, což zapříčiní automatické vyfocení několika snímků, následné zhodnocení a poslání na server, nebo můžeme zapnout tlačítko v levém dolním rohu pro zapnutí logování.

Pokud tedy zatřeseme telefonem, zavolá se metoda `onShake()`, ve které se následně zkontroluje pozice (jak bylo psáno v předchozí kapitole) a poté je vytvořena a spuštěna třída `AsyncTask`, která nám spustí 2 sekundy trvající prodlevu, po které je zavolána metoda `shakeCapture()`, ve které se přidá na mapu nový zájmový bod (Marker) a může začít automatické focení.

Především nás ale zajímá tlačítko vlevo dole, které je realizováno třídou `CompoundButton`. Na toto tlačítko je zaregistrován posluchač události `OnCheckedChangeListener`, který nám hlídá změnu stavu Zap./Vyp. Po zapnutí tlačítka se zkontroluje, zda je zapnuto GPS v zařízení. Pokud není, zobrazí se dialog, který nás o tomto informuje a spustí aktivitu pro nastavení polohy. Pokud je GPS zapnuto, nastaví se zámek CPU pomocí `wakeLock.acquire()`, aby aplikace v průběhu logování neusnula a za pomocí metody `getWindow().addFlags()` se v průběhu také nevypne obrazovka. Nakonec se odebere `coarseLocationListener` pro sledování přibližné polohy a registruje se `fineLocationListener` pro přesnější zjišťování polohy v daném intervalu. Tímto je tedy logování zapnuto. Následně je při každé aktualizaci polohy volána metoda `setFineLocationOnMap(Location location)`, ve které se nastavují informace na spodní panel, aktualizuje se poloha na mapovém podkladu, ale především se nejprve určuje nejbližší zájmový bod ze všech možných uložených bodů, které se načítají z již zmiňované SQLite databáze. Poté se voláním metody `checkDistance(int id)` (kde id je id nejbližšího zájmového bodu) zjišťuje, jak je tento bod vzdálen od zařízení. Po přiblížení tomuto nejbližšímu bodu na vzdálenost menší než 250 metrů se provádějí úkony pro zpřesňování polohy, a to tak, že se odeberou aktualizace z listeneru `fineLocationListener`, přičemž je volána metoda `getAccuracyFineLocation()`, která nám zapříčiní, že aktualizace polohy budou získávány, jak jen to bude možné. Hned poté se do pole `inMarkerRadius[id]` uloží příznak, že zájmový bod je v rádiusu 250 metrů. Po zpřesnění polohy se volá metoda `checkCapture()` pro každý zájmový bod, který je v tomto rádiusu 250 metrů. V této metodě `checkCapture()` se následně zjišťuje, zda je daný bod vzdálen od zařízení alespoň 50 metrů, a zda je kamera zařízení v úhlu zobrazení s tímto bodem. Získání vzdálenosti od bodu, natočení zařízení a jeho výpočet vzhledem k tomuto bodu a nakonec samotnou podmínku lze vidět na následující straně ve výpisu části kódu 4.

```
float distance = myMarkers.get(id).getDistance();
float bearingTo = myMarkers.get(id).getBearing();
float diffBearing = Math.min((bearing - bearingTo + 360) % 360, (bearingTo -
    bearing + 360) % 360);
if (distance < CAPTURE_RADIUS && diffBearing < halfAngleBearing) {
   ...
}
```

Výpis 4: Podmínka pro focení zájmového bodu

V okamžiku splnění obou těchto podmínek se do pole `captureThreads[id]` instanciuje třída `CaptureRunnable`, a v ní se spouští vlákno pro focení daného bodu. Jakmile focení skončí (zařízení je mimo rádius 50 metrů, nebo není v úhlu zobrazení s bodem), ukončí se korespondující vlákno a pro tento bod se volá metoda `startBiqi(capturedPoints)`, která přebírá parametr `capturedPoints`, jenž je kolekcí objektů `MyCapturedPoint` (vyfocených zájmových bodů spolu s informacemi). V této metodě je volán algoritmus BIQI, který kolekci pořízených fotek zhodnotí a určí „nejlepší“. Tato metoda je synchronizována, tudíž nemohou běžet současně 2 tyto metody, a to z důvodu velké paměťové náročnosti algoritmu BIQI. Jakmile je rozhodnuto, která fotka určitého zájmového bodu je nejlepší, volají se tyto metody na ukládání souborů do paměti:

- `saveBin(bestPoint)` - V metodě se ukládají informace v podobě serializovaného objektu `MyCapturedPoint` ve formátu *bin* (pro zpětné načítaní dat informací v galerii), ale také se zde vytváří soubor ve formátu *json*, který obsahuje výběr hlavních informací z objektu pro zaslání souboru na server. Soubory se ukládají do složky *GPSPhotoLogger/bin/*.
- `saveBestImage(bestPoint)` - Metoda, ve které se ukládá nejlepší fotografie ve formátu *jpg*. Ukládá se do složky *GPSPhotoLogger/images/*.
- `saveAllImages(bestPoint, capturedPoints)` - Pro ukládání všech vyfocených fotografií daného zájmového bodu ve formátu *jpg*. Toto je vhodné zejména při testování, zda algoritmus BIQI vybral opravdu „nejlepší“ fotografii z hlediska HVS. Tyto fotky se ukládají do složky *GPSPhotoLogger/images/*.

Názvy dalších podsložek a samotných souborů je tvořen „časovým razítkem“, které je vytvořeno v metodě `getTimeStamp(MyCapturedPoint point)`, dle data vyfocené daného zájmového bodu. Jakmile jsou všechny soubory uloženy, zobrazí se informace, že fotografie byla uložena a soubory jsou připraveny na odeslání na server. Funkcionalita rozhodování, kdy se má poslat fotografie a soubor s informacemi o zájmovém bodu na server řeší třída `BestImageFileObserver`, a to tak, že hlídá zapsání nového souboru do dané složky. Pokud tento soubor najde, pomocí třídy `FileUpload` jej pošle na server. O těchto třídách je psáno v dalších bodech. Po odeslání souborů na server, je přehrán notifikační zvuk a je vytvořena notifikace.

Tato třída je daleko obsáhlejší a disponuje daleko více metodami (pro práci s mapou, zjišťování stavu, zda běží vlákna, zda se fotí, vyhodnocování natočení zařízení apod.). Proto je doporučeno shlédnout zdrojové kódy této aktivity, které jsou uvedeny v příloze A.

7.1.8.2 CoarseLocationListener

Třída pro nalezení přibližné, nebo poslední polohy s rozhraním `OnCoarseLocationListener`, která definuje 3 metody, spouštějící se v závislosti na událostech hledání polohy. Důležitá je ovšem metoda `getCoarseLocation()`, která se v aplikaci volá, pokud chceme zjistit přibližnou polohu. Metoda se prvně pokusí použít poslední platnou pozici, ovšem pokud nevyhovuje nastaveným podmínkám (stáří polohy, přesnost polohy), tak se během 30 sekund snaží najít pozici pomocí mobilní sítě, Wi-Fi, nebo GPS. Pokud se do stanovené doby pozice nenajde, lokace nebude nalezena. Ovšem metodu `getCoarseLocation()` můžeme zavolat z hlavní aktivity `PhotoLoggerActivity` za pomocí stisku tlačítka pro nalezení/přesunu na pozici v pravé dolní části obrazovky. Metoda se volá především po načtení mapy, abychom okamžitě určili polohu zařízení. Třída disponuje ještě metodou `getBetterCoarseLocation()`, která se volá, pokud po zatřesení mobilem není poloha nalezena, nebo není aktuální.

7.1.8.3 FineLocationListener

Obdobná třída jako výše zmiňovaná s tím rozdílem, že `FineLocationListener` hledá pouze přesnou polohu zařízení podle GPS. Třída disponuje metodou `getFineLocation()`, která hledá pozici v intervalu 2 sekundy, nebo pokud se poloha změní o 2 metry. Metoda se volá ihned po zapnutí tlačítka pro logování, nebo pokud se zařízení vzdálí od nejbližšího bodu na více než 250 metrů. Ovšem třída definuje také metodu `getAccuracyFineLocation()`, po jejímž zavolání se bude poloha aktualizovat okamžitě, jak jen to bude možné. Tato metoda se volá, pokud je zařízení blíže, než 250 metrů nejbližšímu zájmovému bodu.

Obě tyto třídy definují již zmiňované rozhraní `OnCoarseLocationListener`, jenž obsahuje 3 metody, které jsou zobrazeny v následujícím výpise 5.

```
public interface OnCoarseLocationListener {
   void onLocationStart();                  // Run, when start finding location
   void onLocationFind();                   // Run, when location is find
   void onLocationSet(Location location);  // Run, when can set location
}
```

Výpis 5: Rozhraní pro události polohy

7.1.8.4 MovementListener

Třída implementující rozhraní `SensorEventListener`. Třída taktéž obsahuje vytvořené rozhraní `OnMoveListener`, které obsahuje metody, jenž se volají při určitých událostech senzorů. Tato třída nám tudíž zjišťuje stav senzorů zařízení. Konkrétně zjišťuje stav ze dvou typů senzorů:

- Senzor `ROTATION_VECTOR`

  Typ senzoru `ROTATION_VECTOR` je kombinací tří senzorů pro sběr dat. Skládá se ze *senzoru magnetického pole*, z *gyroskopu* a z *akcelerometru*. Pomocí tohoto typu senzoru se určuje natočení přístroje vzhledem k magnetickému severu. Využití tohoto senzoru je tedy nasnadě. Aktualizace probíhá v metodě `onSensorChanged()` pokaždé, když se změní data senzoru. Výstupem je rozsah hodnot od -180°do +180°, kde 0°znamená, že je zařízení natočeno přímo na magnetický sever, záporné hodnoty ukazují směrem na západ a kladné směrem na východ. Tato hodnota se předává v metodě `onRotation(bearing)`, kde `bearing` je ono zmiňované natočení. Tato hodnota nám ale ukazuje natočení vůči magnetickému severu, což není pro aplikaci příhodné. Přímo v aktivitě `PhotoLoggerActivity` se proto volá metoda `getDeclination()`, která nám vrátí *deklinaci* (úhlovou vzdálenost) mezi magnetickým a pravým severem. Tuto hodnotu poté stačí přičíst k hodnotě proměnné `bearing` a výsledná hodnota značí natočení vůči pravému severu, kterou v aplikaci potřebujeme. Toto natočení nám poslouží především přímo pro změnu pohledu na mapovém podkladu a také pro výpočet úhlu, v jakém se zařízení nachází vůči focenému zájmovému bodu. V následujícím výpisu kódu 6 lze vidět způsob výpočtu úhlu natočení vůči magnetickému severu v události změny dat senzoru `ROTATION_VECTOR`. Po získání dat se nejprve musí přemapovat osa koordinačního prostoru senzoru (osa Y je mapována osu Z). Poté se vypočte hodnota azimutu (rotace kolem osy -Z) a nakonec se tato hodnota pomocí metody `Math.toDegrees(double angrad)` přepočte na stupně.

```
if (event.sensor.getType() == Sensor.TYPE_ROTATION_VECTOR) {
   SensorManager.getRotationMatrixFromVector(rotationMatrix,event.values);
   SensorManager.remapCoordinateSystem(rotationMatrix, SensorManager.
       AXIS_X, SensorManager.AXIS_Z, remapRotationMatrix);
   float[] orientation = new float[3];
   SensorManager.getOrientation(remapRotationMatrix, orientation);
   float bearing = (float) Math.toDegrees(orientation[0]);
   listener.onRotation(bearing);
}
```

Výpis 6: Výpočet úhlu natočení zařízení vůči magnetickému severu

- Senzor `ACCELEROMETER`

  Akcelerometr je senzor měřící zrychlení (akceleraci) zařízení. Pomocí něj je tedy zjišťován stav, zda je se zařízením zatřeseno, či nikoli. Tato hodnota zrychlení se spočte ze všech tří os senzoru, aby výstupem byl výsledek, který značí hodnotu gravitačního zrychlení (tedy 1g = gravitační síla působící na Zemi $\approx 9{,}8\ m/s^2$). Hodnota, při jejímž překročení se spustí metoda `onShake()` je ve výchozím stavu nastavena na 10g. Ve zjišťování stavu senzoru je také počítáno s tím, že se událost spustí nejdříve po 3 sekundách od minulého zatřesení.

7.1.8.5 CameraPreview

Třída, která má na starosti zobrazení náhledu výstupu integrované kamery mobilního zařízení v pravém horním rohu obrazovky. Prvek, ve kterém je zobrazen tento náhled, je řešen třídou `SurfaceView`. Zajímavá je především funkce změny rozměru tohoto prvku implementované pomocí `View.OnTouchListener`, kde se pomocí odchycení pohybu po tomto prvku zvětšuje (pohybem doleva) nebo zmenšuje (pohybem doprava) rozměr tohoto prvku v závislosti na rozlišení a PPI obrazovky zařízení. Stav rozměru se při zrušení prvku `SurfaceView` ukládá do *SharedPreferences* a samozřejmě se z tohoto úložiště opětovně načítá v konstruktoru této třídy. Klíčové příkazy pro nastavení rozměru prvku `SurfaceView` lze vidět v následujícím výpise 7.

```
density = getResources().getDisplayMetrics().density;
params = this.surfaceView.getLayoutParams();
params.width = getFromPrefs(getContext(), PREFS_VIEW_WIDTH, (int) (
    DEFAULT_WIDTH * density));
params.height = getFromPrefs(getContext(), PREFS_VIEW_HEIGHT, (int) (
    DEFAULT_HEIGHT * density));
this.surfaceView.setLayoutParams(params);
```

Výpis 7: Nastavení rozměru prvku SurfaceView

Další důležitou funkcí, kterou je třeba zmínit je ta, že prvek `SurfaceView` je překryt dalším prvkem `SurfaceView` (průhledným), který změní barvu na červenou (z části průhlednou), když je právě pořizována fotografie. Tato funkce je řešena pomocí třídy `Handler`, která definuje metodu `handleMessage(Message inputMessage)`, jenž je volána právě při změně stavu kamery ze třídy `CaptureRunnable`, o které se píše dále.

7.1.8.6 CaptureRunnable

Ve stručnosti třída `CaptureRunnable` slouží k automatickému focení a následnému bufferování těchto snímků. Třída implementuje rozhraní `Runnable`, které obsahuje metodu `run()`. Jde tedy o třídu, po jejíž inicializaci běží další kód v samostatném vlákně. Toto vlákno se spouští pro každý zájmový bod samostatně ve chvíli, kdy je zařízení od daného bodu vzdáleno méně než 50 metrů. Vlákno se taktéž spouští po zatřesení mobilním telefonem.

V konstruktoru třídy se nejprve do statické proměnné `maxImagesOfPoint` z nastavení načte maximální počet snímků na jeden zájmový bod. Tato hodnota je ve výchozím stavu nastavena na 10 snímků na bod. Po vytvoření instance této třídy se může spustit metoda `run()`. Po spuštění vlákna se tedy provádí focení pomocí metody `takePicture(null, null, jpeg)`, přičemž se také zasílá stav, že zařízení fotí. Po pořízení snímků je volán `Callback PictureCallback jpeg`, kde je zasílán stav, že kamera přestala fotit a je spuštěna metoda `addDataToCache(data)`, která bufferuje pořízené snímky. Zde se vytvoří již zmiňovaný objekt `MyCapturedPoint` s daty: *fotografie*, *zájmový bod* v podobě reference třídy `Marker`, *současná poloha zařízení* (odkud se bod fotí), *datum*, *vzdálenost od bodu* a *natočení zařízení* vzhledem k bodu a vzhledem k severu.

Pro každou fotku se tedy vytvoří jeden objekt třídy `MyCapturedPoint`, který se následně ukládá do generické kolekce typu `List`. Jelikož ale budeme uchovávat maximálně počet snímků, jenž je uložen v proměnné `maxImagesOfPoint`, musí se tyto data bufferovat, neboli uchovávat pouze tento maximální daný počet snímků. Zde nastává otázka, které body uchovávat. Aplikace může na dané poloze vyfotit třeba 50, 100 snímků apod. Následuje tedy algoritmus, který by měl vybrat nejoptimálnější snímky v závislosti na poloze a natočení zařízení vzhledem k zájmovému bodu. Tento algoritmus najde index v kolekci `List`, kde je uložen „nejhorší“ bod, dle podmínek (vzdálenost a natočení od bodu), které jsou zobrazeny v následujícím výpise 8.

```
for (int i = 0; i < capturedPoints.size(); i++) {
   distance = capturedPoints.get(i).getDistance();
   bearing = capturedPoints.get(i).getBearing();
   if (Math.abs(bearing) > PhotoLoggerActivity.ANGLE_OF_VIEW / 2) {
      multiplier = 1.5f;
   } else if (Math.abs(bearing) > PhotoLoggerActivity.ANGLE_OF_VIEW) {
      multiplier = 2.0f;
   }
   ratio = distance * multiplier;
   ratios[i] = ratio;
}
for (int i = 0; i < ratios.length; i++) {
   if (ratios[i] > maxValue) {
      maxValue = ratios[i];
      maxIndex = i;
   }
}
```

Výpis 8: Algoritmus pro určení snímku zájmového bodu pro odstranění z kolekce

Poté se pomocí nalezeného indexu tento bod odebere a nyní je v kolekci `List` uložen maximálně nastavený počet snímků jednoho zájmového bodu, který může být dále předán k hodnocení algoritmu BIQI.

Ovšem ve třídě je ještě důležité zmínit jednu podstatnou věc. Jelikož focení zájmového bodu je spouštěno ve vláknech, mohou tyto vlákna pro focení běžet současně. Zde nastává problém, jelikož k metodě `takePicture(null, null, jpeg)` nemůže přistupovat vyšší počet vláken zároveň (obzvlášť v době, kdy se zpracovávají vstupní data ze senzoru kamery). Proto je implementována třída `MyLock` k synchronizaci těchto vláken. Tudíž před každým započetím focení fotografie se musí vlákno uzamknout pomocí synchronizované metody `lock()` a po dokončení focení se vlákno opět odemkne pomocí metody `unlock()`.

7.1.8.7 MyCapturedPoint

Nyní je potřeba si něco říci o již několikrát zmiňované třídě `MyCapturedPoint`. Již je nám známo, že tato třída je použita napříč celou aplikací, jelikož je velmi důležitá a reprezentuje nám objekt právě jednoho zájmového bodu. V tomto objektu je uložena především vyfocená fotografie onoho zájmového bodu, ale také další informace s tímto bodem spojené. Třída `MyCapturedPoint` obsahuje tedy tyto členy:

- „Nejlepší“ fotografie zájmového bodu (`byte[]`)
- Jméno, popis a jméno pro ukládaný soubor zájmového bodu (`String`)
- Poloha zájmového bodu a poloha zařízení v době focení (`Double`)
- Datum vyfocení zájmového bodu (`Date`)
- Vzdálenost od zájmového bodu (`Float`)
- Natočení zařízení vzhledem k severu a vzhledem k bodu (`Float`)
- Hodnocení fotografie algoritmem BIQI (`Double`)

Tato třída implementuje rozhraní `Serializable`, tudíž nám umožňuje uložit objekt do formátu, vhodného pro uložení do paměti zařízení. Tohoto je zejména využito pro uložení nejlepšího zájmového bodu a opětovného načtení souboru pro zobrazení informací v galerii. Pro posílání těchto informací na server jsou členy třídy zapsány do formátu JSON s využitím třídy `JSONObject`, a tento objekt je následně zasílán na server (spolu s fotografií).

7.1.8.8 FileUpload

Třída uložena v balíčku `data`, která slouží pro posílání fotografie a příslušných informací na server. Tato třída rozšiřuje třídu `AsyncTask`, tudíž proces posílání souborů na server probíhá v samostatném vlákně. Důležitý je zde konstruktor třídy, který přebírá parametry: referenci třídy `PhotoLoggerActivity`, id uživatele, název a souborový proud fotografie a také název a souborový proud JSON objektu. V tomto konstruktoru se tedy inicializují všechna tato data a nastavuje se URL pro nahrávání souborů. Po této inicializaci se může proces nahrávání souborů spustit voláním metody `execute()` vytvořeného objektu. Jelikož jde o vlákno `AsyncTask`, proces nahrávání souborů probíhá v metodě `doInBackground(Void... params)` a to následovně:
Nejprve je vytvořena instance třídy `HttpsURLConnection`, která otevře spojení se serverem, přičemž se povolí přijmutí všech certifikátů (kvůli Self-signed certifikátu na serveru), jelikož spojení je realizováno protokolem HTTPS. Následně se nastaví časový interval 10 sekund pro spojení a je vybrána POST dotazovací metoda pro nahrávání souborů na server. Metoda POST je nastavena jako typ *multipart/form-data*, můžeme tedy do dat metody jednoduše přidat datové proudy fotografie a JSON objektu. Nakonec je tento dotaz s daty poslán na server, kde jsou data následně zpracována.

7.1.8.9 BestImageFileObserver

Třída rozšiřující abstraktní třídu `FileObserver`, která monitoruje změny v souborovém systému zařízení. Konkrétně v našem případě se sleduje událost typu vytvoření souboru. Ihned v konstruktoru třídy je nastavena cesta (*GPSPhotoLogger/images/*), která se má sledovat. Při vyvolání události typu `CREATE` se zavolá metoda `onEvent(int event, String path)`. V této metodě se tedy pomocí proměnné `path` zjistí název právě uložené fotografie a souboru JSON, a je také zjištěno ID uživatele ze statické třídy `UserID`. Díky těmto informacím se může instanciovat již zmiňovaná třída `FileUpload` pro nahrávání těchto souborů na server. Tento objekt je následně přidán do fronty `queue`, která je implementována strukturou spojového seznamu (`LinkedList`) a je zavolána metoda `startSendRunnable()`. Tato metoda spustí nové vlákno `sendRunnable` (pokud již neběží) a následně za pomocí třídy `Handler` je kód v tomto vlákně volán každou sekundu. V tomto vlákně se nejprve kontroluje obsah fronty. Pokud fronta obsahuje alespoň jeden objekt `FileUpload`, pomocí statické proměnné `boolean isFastConnect` se následně kontroluje, zda je zařízení připojeno k internetu a jaký je typ tohoto připojení. Pokud je toto připojení dostatečně rychlé, smaže se první objekt `FileUpload` z fronty `queue` a následně se pomocí metody `execute()` spouští vlákno `AsyncTask` ve třídě `FileUpload` pro nahrání těchto souborů na server. Ovšem pokud není připojení k internetu k dispozici nebo je připojení pomalé, soubory se na server neodešlou, objekt je ponechán ve frontě a v intervalu 1 sekundy se toto připojení opět kontroluje.

Abychom mohli zjistit aktuální stav a typ připojení k internetu, je v konstruktoru této třídy zaregistrován `BroadcastReceiver` s Intent filtrem `CONNECTIVITY_ACTION`. Tato komponenta s názvem `connectionReceiver` nám tedy umožňuje naslouchání oznámení a reaguje tedy právě na změnu připojení k internetu. Při této změně je volána metoda `onReceive(...)`, jejíž obsah lze vidět ve výpise 9 a je zde zjišťován právě stav a typ připojení. Pokud je připojení k internetu aktivní, volá se metoda `isConnectionFast(int type, int subType)`, pomocí níž je zjišťován typ připojení a na základě tohoto typu je rozhodnuto, zda je připojení dostatečně rychlé, či nikoli. Výsledek, zda je připojení dostačující je nakonec zapsán do proměnné `isFastConnect`.

```
NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
int networkSubType = telephonyManager.getNetworkType();

if (networkInfo != null && networkInfo.isConnected()) {
   isFastConnect = isConnectionFast(networkInfo.getType(), networkSubType);
} else {
   isFastConnect = false;
}
```

Výpis 9: Zjištění stavu a typu připojení

7.1.9 Algoritmus BIQI

Nyní je nasnadě popsat princip algoritmu BIQI z hlediska implementace, jelikož popis algoritmu byl již uveden v podkapitole 4.2.4. Prvně je si třeba říci, že celý proces hodnocení obrázků probíhá samozřejmě v samostatném vlákně. Toto vlákno je vytvořeno ihned po vyfocení snímků zájmového bodu a je volána metoda `startBiqi(final MyArrayList capturedPoints)`, jenž je synchronizovaná z důvodů velké paměťové náročnosti algoritmu BIQI. Pro samotné hodnocení fotografie je v metodě následně volána statická metoda `CalcQuality(bitmap)` ve třídě `Biqi`.

Tato třída je tedy pro algoritmus BIQI klíčová. Volaná metoda `CalcQuality(Bitmap bmp)` přebírá parametr `Bitmap`, jenž je testovaným obrazem. V této metodě se nejprve zavolá metoda `toGrayscaleArray(Bitmap bmp)` pro převod obrazu do stupňů šedi, kde je obraz rovnou převeden do 2D pole reálných hodnot, jelikož s tímto polem algoritmus pracuje. Ihned poté jsou instanciovány třídy `Dwt` s Daubechies 9/7 vlnkou (která řeší funkčnost diskrétní 2D vlnkové transformace) a vnitřní statická třída `Dwt2Coefs` pro ukládání koeficientů z transformace. Následně je tedy provedena diskrétní vlnková transformace (DWT) pro testovaný obraz. Výsledek transformace jsou tedy 4 koeficienty: **cA** (aproximovaný koeficient z dolní propusti), **cH**, **cV**, **cD** (detailní koeficienty z horní propusti v každém směru). Tyto 3 detailní koeficienty jsou následně parametrizovány zobecněnou Gaussovou distribucí (GGD), což řeší třída s názvem `Ggd`. K dalšímu zvýšení frekvenčního rozlišení se tato transformace ještě 2× opakuje, přičemž je jako vstup DWT brán koeficient **cA**, který vznikl z minulé transformace. Po provedení tohoto rozkladu ve třech stupních máme k dispozici 18 hodnot (9 hodnot σ a 9 hodnot γ) uložených v 1D poli s názvem `rep`, které je následně předáváno SVM pro klasifikaci. Ovšem toto pole musí být nejprve pomocí metody `saveValuesToFile(double[] rep)` uloženo do paměti zařízení (konkrétně do *GPSPhotoLogger/libsvm/*), jelikož zmiňovaná knihovna *libsvm*, ale především 2 podpůrné třídy `svm_predict` a `svm_scale` pracují pouze s uloženými textovými soubory.

Následně tedy může začít hodnocení obrazu. Prvně se klasifikuje rušení v obraze pomocí metody `launchSvmScale(String dis)` s vybraným DIS modelem a poté může proběhnout měření pravděpodobnosti výskytu daných rušení za pomocí `launchSvmPredict(String dis, ...)`. Tyto pravděpodobnosti výskytu rušení se uloží do souboru. Poté se provádí samotné měření skóre (míry rušení) opět pomocí těchto dvou metod, kdy jsou prvně data škálována s korespondujícím rozsahem DIS modelů a poté jsou data s tímto DIS modelem pomocí SVM srovnávána (kromě určení kvality JPEG, která je řešena samostatně ve třídě `JpegQualityScore`). Z tohoto tedy získáme míru daného rušení v obraze. Všechny tyto DIS modely jsou uloženy ve složce *GPSPhotoLogger/libsvm/*. Nakonec jsou všechny pravděpodobnosti výskytu rušení vynásobeny s korespondující mírou rušení, poté sečteny a nakonec je získáno výsledné skóre obrazu. Metoda `CalcQuality(Bitmap bmp)` vrací pole reálných hodnot `ret`, kde se na první pozici vyskytuje skóre kvality a následují skóre daných rušení (JPEG, JP2K, WN, GB, FF).

Poznámka 3 Algoritmus BIQI jsem přepisoval do *Javy* ze zdrojového kódu *Matlabu*, který je k dispozici na: `http://live.ece.utexas.edu/research/quality/BIQI_release.zip`

7.2 Serverová část - webová aplikace

Nedílnou součástí celého systému je také samozřejmě aplikace, která by plnila funkčnost na straně serveru. Tato aplikace byla vyvíjena v prostředí programu *Visual Studio 2013*, což je IDE od společnosti Microsoft pro vývoj konzolových, desktopových, webových aplikací atp., převážně na platformě .NET. Pro tuto práci jsem tedy potřeboval vytvořit ASP.NET webovou aplikaci založenou na technologii .NET a naprogramovanou v jazyce C#. V této webové aplikaci jsem použil prvky JavaScriptu a také webové služby, o kterých bude psáno dále. Poté bylo zapotřebí vytvořit již zmiňovanou databázi v prostředí *Microsoft SQL Server 2014*. Obě tyto komponenty (webová aplikace i databáze) tedy musejí být přístupné na nějakém webovém serveru. Pro tuto práci jsem vybral druhý nejpoužívanější webový server, který se nazývá IIS (Internet Information Services), nebo také Internetové Informační Služby, taktéž od společnosti Microsoft pro operační systémy Windows. Zdrojové kódy webové aplikace lze nalézt v příloze A.

7.2.1 Webový server

Na samotném začátku vytváření serveru jsem musel rozhodnout na jakém OS tento webový server poběží. Na výběr jsem měl z několika možností, ale nakonec jsem si vybral platformu Windows, konkrétně operační systém *Windows Server 2012 R2*, což je OS z řady Windows NT od firmy Microsoft. Poté jsem musel vybrat webový server, který na tomto stroji poběží. V tomto případě bylo rozhodnutí jednoduché, jelikož součástí *Windows Server 2012 R2* je již zmiňovaný webový server IIS ve verzi 8.5. Nyní bylo zapotřebí tento webový server nakonfigurovat. Po instalaci onoho IIS se nainstaluje také program *Správce Internetové informační služby*, ve kterém se provádí veškeré konfigurace tohoto softwarového webového serveru. Prvně jsem vytvořil web s názvem *GPSPhotoLogger* a nastavil port, na kterém bude web přístupný na číslo 443, což je protokol HTTPS (Šifrovaný přenos HTTP protokolu přes TLS). Následně jsem také vytvořil web přístupný přes port 80 (HTTP), který pouze uživatele přesměruje na web *GPSPhotoLogger*. Jelikož protokol HTTPS je založen na symetrickém šifrování, musí být na straně serveru digitálně podepsaný veřejný klíč, tedy certifikát. Z důvodu nemožnosti získat tento certifikát podepsaný certifikační autoritou, jsem byl nucen vygenerovat certifikát podepsaný sám sebou (Self-signed certifikát). Tento certifikát byl následně uložen na straně serveru v úložišti certifikátů a poté byl přidružen k webu *GPSPhotoLogger*. Po nastavení přidruženého fondu aplikací a různých dalších méně podstatných nastavení jsem mohl vložit již vytvořený web do předem nastavené složky, a tímto byl celý proces nastavování hotov a webový server byl připraven k provozu.

7.2.2 Databáze

Dalším krokem bylo nainstalovat instanci *Microsoft SQL Server 2014* na tento server a následně v něm vytvořit tabulky, které jsou již popsány v podkapitole 6.3.3. Celý postup konfigurace a vytvoření tabulek je popsán na následující straně.

Po instalaci databázového systému *Microsoft SQL Server 2014* se také nainstaloval program pro konfiguraci této databáze s názvem *Microsoft SQL Server Management Studio*, kde jsem musel nejprve vytvořit novou databázi. Tu jsem pojmenoval příhodným jménem: *Database*. Po vytvoření této databáze již stačilo vytvořit databázové tabulky pro uchování uživatele a jeho zájmových bodů. Příkazy pro vytvoření těchto tabulek jsou uvedeny v následujícím výpise.

```
CREATE TABLE [dbo].[User] (
    [ID] INT IDENTITY (1, 1) NOT NULL,
    [Name] VARCHAR (50) NOT NULL,
    [Email] VARCHAR (50) NOT NULL,
    [Password] VARCHAR (40) NOT NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC),
    UNIQUE NONCLUSTERED ([Email] ASC)
);

CREATE TABLE [dbo].[Marker] (
    [ID] INT IDENTITY (1, 1) NOT NULL,
    [UserID] INT NOT NULL,
    [Name] VARCHAR (50) NOT NULL,
    [Description] VARCHAR (100) NOT NULL,
    [Latitude] FLOAT (53) NOT NULL,
    [Longitude] FLOAT (53) NOT NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Výpis 10: SQL příkazy pro vytvoření databázových tabulek

Avšak po vytvoření tabulek bylo zapotřebí zajistit, aby se k těmto tabulkám mohli uživatelé aplikace dostat, tedy aby k nim měli přístup. Následovalo tedy vytvoření databázového uživatele s názvem: *user* s přidruženým heslem: *password*. Tomuto uživateli byla nastavena výchozí databáze: *Database* a následovalo přidělení práv k vytvořeným tabulkám a to následovně:

- Tabulka `Marker` - `INSERT`, `SELECT`, `DELETE`, `UPDATE`
- Tabulka `User` - `INSERT`, `SELECT`, `UPDATE`

Z obou aplikací (klientské i serverové) se tedy k databázi přistupuje pouze za pomocí údajů tohoto uživatele s výše danými právy. Tímto je zamezeno přístupu k dalším případným databázovým tabulkám apod. Jelikož se klientská aplikace pro Android připojuje k databázi přímo, bylo ještě zapotřebí přidat výjimku ve Windows firewall jako příchozí pravidlo, které povoluje všechna připojení na portu 1433 (číslo portu pro připojení k *Microsoft SQL Server* databázi).

7.2.3 Webová aplikace

Již bylo řečeno, že aplikace, sloužící jako serverová komponenta byla implementována za použití jazyka C# s využitím technologie ASP.NET. Nyní je nasnadě si něco říci o samotné implementaci této aplikace. Celá koncepce webových stránek je složena ze dvou částí:

- Vizuální prvky stránky, kam patří značky, serverové ovládací prvky, statický text, ale především kód napsaný v jazyce JavaScript, který se spouští až po stažení stránky (tzv. na straně klienta). Tohoto kódu je např. využito při zobrazování mapového podkladu, přidávání zájmových bodů, apod.
- Programová logika stránky, kam patří obslužné rutiny událostí, ale především kód, který instancuje třídy a volá jejich metody. Tyto třídy a metody jsou uloženy na straně serveru ve složce `App_Code`.

V aplikaci je také využito webových služeb, což je jednoduše řečeno softwarový systém umožňující interakci dvou strojů na síti. Pod tímto si lze typicky představit stroj klienta komunikující se strojem, který plní funkci serveru. Tohoto systému je především využito při volání členských metod tříd na serverové straně z klientské strany z JavaScriptového kódu. Systém zejména plní funkce načítaní, přidávání a mazání zájmových bodů uživatele z databáze, aniž by se stránka opětovně načítala.

Pro částečné pochopení struktury aplikace je zde shrnuta pouze důležitá část struktury projektu webové aplikace. V kořenovém adresáři se nalézá především hlavní stránka s přihlášením, ale také stránka pro nahrávání souborů z klientské aplikace. Následuje rozdělení do složek:

- `App_Code`
  - třídy `Database` a `DbTable` - pro připojení k Microsoft SQL databázi
  - třídy `Marker` a `MarkerTable` - ORM k databázové tabulce zájmového bodu
  - třídy `User` a `UserTable` - ORM k databázové tabulce uživatele
  - webová služba `WebService` - pro komunikaci mezi klientem a serverem
- `User` - Obsahuje stránku s mapovým podkladem pro přidávání zájmových bodů, i stránku pro prohlížení galerie, kde mají přístup pouze přihlášení uživatelé
  - `Data` - pro ukládání fotografií a informací zájmového bodu daného uživatele

V aplikaci je také použita knihovna ***Json.NET-8.0.2***, která slouží pro parsování informací z přijatého JSON souboru [1]. V následujících bodech je popsána implementace všech webových stránek v aplikaci.

[1] Ke stažení na: `http://www.newtonsoft.com/json`

7.2.3.1 Stránka pro přihlášení uživatele

Úvodní stránka s názvem `Default.aspx`, která se zobrazí ihned po navštívení dané adresy serveru. Na stránce je zobrazeno logo (lze vidět na obrázku 21) a název aplikace. Pod ním je odkaz na stažení aplikace, ale především formulář pro vyplnění e-mailu a hesla. Tyto stránky tedy plní jedinou jednoduchou funkcionalitu, a to přihlášení uživatele.

Obrázek 21: Logo aplikace

Po kliknutí na tlačítko „Přihlásit“ se pouze vytvoří instance třídy `UserTable` a zavolá se metoda `CheckCredentials(String email, String password)`. V této metodě se aplikace připojuje k databázi a ověřuje se shodnost zadaných údajů. Metoda vrací výčtový typ s názvem `Login`, který udává, zda jsou údaje správné, nebo nastala nějaká chyba.

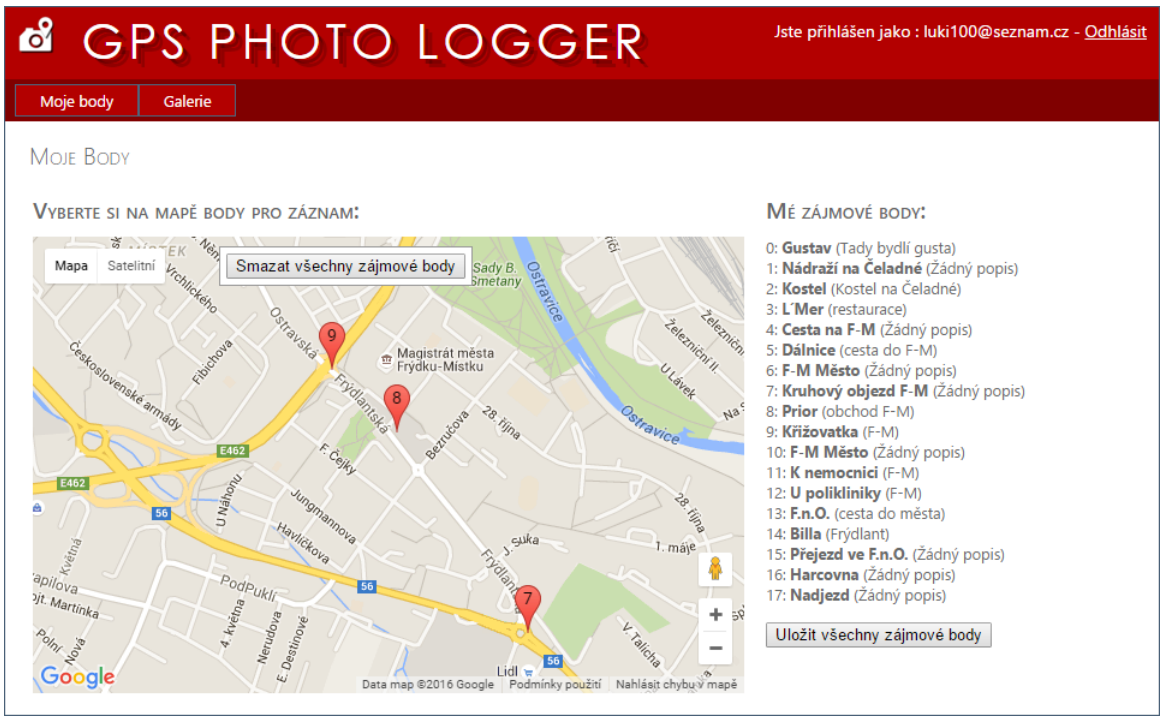
7.2.3.2 Stránka pro přidávání zájmových bodů

Po úspěšném přihlášení se uživateli zobrazí stránka s mapovým podkladem pro přidání zájmových bodů, která je uložena ve složce `Data` a nese název `Default.aspx`. Tuto stránku lze vidět na následující straně na obrázku 22. Ačkoli je stránka psaná v ASP.NET, její převážná funkcionalita spočívá v JavaScriptovém kódu. Ihned po načtení stránky se tedy načítá skript, který tvoří *Google Maps JavaScript API*, což je rozhraní v jazyce JavaScript pro práci s Google mapami. Po načtení tohoto API je volána funkce `InitializeMap()`, kde se inicializuje mapový podklad, ale také se volá funkce `doGeolocation()` pro nalezení polohy PC, resp. zařízení na kterém si klient prohlíží tuto stránku. Ve funkci `InitializeMap()` je ale také registrován listener pro odchytávání kliknutí na mapu a je také volána funkce `getAllMarkers()`, která pomocí metody WS (Webové Služby) načte všechny zájmové body právě přihlášeného uživatele. Tuto metodu lze vidět v následujícím výpise.

```
[WebMethod]
public object[] GetAllMarkers()
{
   MarkerTable markerTable = new MarkerTable();
   List<Marker> List = markerTable.getMarkers(Context.User.Identity.Name);
   Object[] obj = (Object[])List.ToArray();
   return obj;
}
```

Výpis 11: WS metoda pro načtení všech zájmových bodů uživatele

Tato metoda tedy pomocí třídy `MarkerTable` a metody `getMarkers(string userName)` vrátí kolekci objektů `Marker`, která je následně přetypována na pole objektů, aby se mohla předat zpět JavaScriptovému kódu, který běží na klientské straně. Po úspěšném vykonání WS metody se tedy vrátí pole objektů a je volána funkce `addMarker(...)` pro každý tento objekt, která přidá tento zájmový bod na mapu. Nakonec je také volána funkce `showMarkers()`, která nám přidá tento zájmový bod v textové podobě do pravé části stránky.



Obrázek 22: Stránka pro přidávání zájmových bodů

Stěžejním bodem této stránky je ale přidávání nových zájmových bodů na mapu. Toto je řešeno přidáním již zmíněného listeneru na mapový podklad, který naslouchá kliknutí na mapě. Po kliknutí na dané místo na mapě se zobrazí dialog vyzývající vepsání názvu zájmového bodu a popisu tohoto bodu odděleného středníkem. Po zadání údajů a následném potvrzení je opět volána funkce `addMarker()` pro přidání zájmového bodu na mapu a taktéž na vepsání tohoto bodu do pravé části stránky. Ovšem pokud chceme tento nově přidaný bod uložit, je zapotřebí kliknout na tlačítko „Uložit všechny zájmové body“, na kterém je volána funkce `saveAllMarkers()`, která opět komunikuje s WS, jenž zasílá data objektu `MarkerTable` ukládající tyto nové zájmové body do databáze. Na stránce je také samozřejmě řešeno mazání všech těchto bodů, a to pomocí tlačítka v horní části mapového podkladu. Kliknutím na toto tlačítko je volána funkce `deleteAllMarkers()`, která opět komunikuje s WS, a ta předává data objektu `MarkerTable` mazající všechny body přihlášeného uživatele.

7.2.3.3 Stránka pro nahrávání souborů na server

Stránka s názvem `FileUpload.aspx` řešící funkcionalitu přijmutí a ukládání souborů na server. Celá tato funkce je řešena v události `Page_Init(...)`, která je volána při inicializaci stránky, tudíž když se uživatel dotazuje na tuto stránku. V této události se tedy nejprve zjišťuje, zda požadavek klienta obsahuje POST data. Pokud tento požadavek obsahuje neprázdnou proměnnou `userID`, může začít ukládání souborů. Pomocí proměnných `userID`, `imgFileName` a `jsonFileName` se tedy určí cesty ukládáných souborů a za využití třídy `HttpFileCollection` a metody `SaveAs(string filename)` se tyto soubory na server uloží. Ovšem pokud požadavek neobsahuje proměnnou `userID`, nebo je tato proměnná prázdná, je klient přesměrován na stránku `Default.aspx` a žádné nahrávání souboru neproběhne.

7.2.3.4 Stránka pro prohlížení fotografií

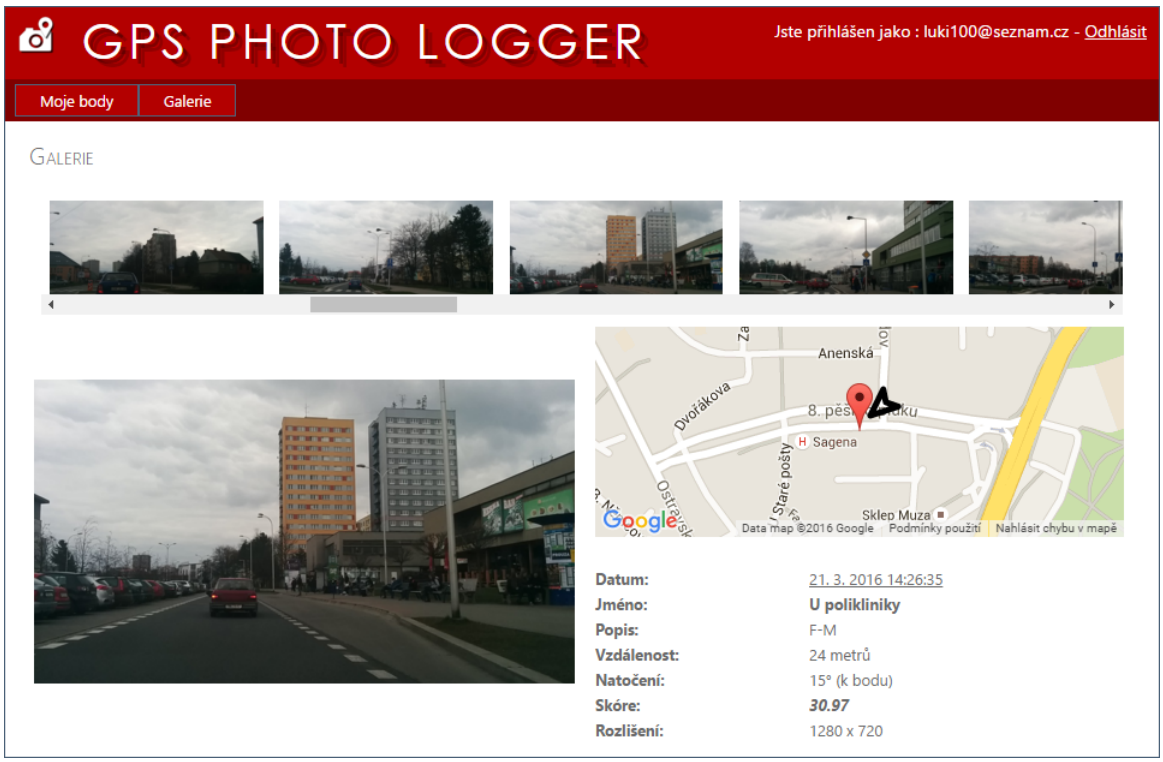
Posledním, ale velmi klíčovým bodem aplikace bylo vytvořit stránku, která by sloužila jako galerie pro prohlížení fotek pořízených klientskou aplikací pro Android. Tato stránka se nachází ve složce `User` pod jménem `Gallery.aspx`. Ke stránce bude tedy mít přístup taky pouze jen přihlášený uživatel. Na stránce je taktéž využito *Google Maps JavaScript API* pro práci s mapou.

V události `Page_Load(object sender, EventArgs e)`, která je volána ihned po načtení stránky se volá metoda `PopulateImages()`. V této metodě se nejprve do řetězce `userId` uloží hodnota id uživatele, která je již uložena v `Session` s názvem `id`. Tato `Session` se vytvořila ihned po přihlášení uživatele do webového rozhraní. Díky tomuto id se může následně zjistit cesta, kde má uživatel na serveru uložené zaslané fotografie. Po zjištění této cesty se nám uloží všechny tyto fotografie do pole tříd `FileInfo`, které se následně seřadí od nejnovějšího data pořízení. Nakonec se toto pole projde, pro každý prvek se vytvoří objekt `MyImages`, který je následně svázán s komponentou `Repeater`. Kód této funkce je vyobrazen v následujícím výpise.

```
string userId = (string)(Session["id"]);
List<MyImages> myImages = new List<MyImages>();
DirectoryInfo dir = new DirectoryInfo(Server.MapPath("~/User/Data/" + userId +
    "/images/"));
FileInfo[] files = dir.GetFiles().OrderByDescending(p => p.CreationTime).
    ToArray();
foreach (var file in files)
{
   myImages.Add(new MyImages { FileName = file.Name });
}
Repeater1.DataSource = myImages;
Repeater1.DataBind();
```

Výpis 12: Načtení fotografií přihlášeného uživatele na serveru

Toto nám tedy umožní zobrazení veškerých uložených fotek v komponentě `Repeater`, se kterou můžeme v horizontální poloze pohybovat pro zobrazení všech fotek. Po kliknutí na vybranou fotografii se nám zobrazí větší náhled fotografie v levé části stránky, a v pravé části stránky se nám zobrazí poloha zájmového bodu na mapě, k tomu také poloha zařízení a natočení tohoto zařízení v době vyfocení bodu. Pod touto mapou se zobrazí další informace o vyfocením zájmovém bodu. Celou tuto stránku lze vidět na následujícím obrázku 23.



Obrázek 23: Stránka pro prohlížení fotografií

Tohoto je dosaženo webovým aplikačním frameworkem *jQuery*, jenž je vlastně JavaScriptová knihovna. Funkce, která nám odchycuje kliknutí na fotku, zjišťuje stav atributu `alt`, ve kterém je uložen název souboru této fotografie a pomocí níž můžeme načíst tuto fotografii s přidruženým JSON souborem. Pro načtení tohoto souboru do webové stránky je opět použito WS, konkrétně metody `getJsonData(int userId, string fileName)`, která nám podle id a názvu souboru zjistí umístění souboru. Poté je tento soubor pomocí tříd `StreamReader` a `JsonConvert` deserializován do objektu s názvem `point`, jenž je instancí třídy `MyCapturedPoint`, obdobně jako v klientské aplikaci pro Android. Jakmile je tento soubor upraven, ihned poté je přetypován na obecný typ `Object`, aby mohl byt navrácen JavaScriptovému kódu, který tento objekt parsuje a vypisuje tyto informace do příslušných elementů stránky. Nakonec se z těchto informací převezmou data již dvou zmiňovaných poloh a nastaví se na mapový podklad.

8 Testování aplikace

Na samotném konci vývoje se aplikace ve většině případů také důkladně testuje. Tímto se zamezí chybám a nedostatkům, které mohly vzniknout při implementaci mé aplikace *GPSPhotoLogger*. Klientskou aplikaci jsem tedy testoval z několika aspektů, a to např.: její plynulost a stabilitu, spotřebu energie, zda se aplikace sama neukončuje apod. Ovšem u vývoje aplikací pro Android je klíčovým testem aplikaci odzkoušet na několika různých reálných popř. virtuálních zařízeních, a to z důvodů, že různá zařízení mohou disponovat různými verzemi OS, ale také mohou mít odlišnou nadstavbu tohoto OS. Z těchto důvodů se tedy testovaná aplikace na různých zařízeních nemusí chovat úplně stejně, nebo nemusí mít stejně rozvrženy grafické prvky, a to z důvodů rozdílných rozlišení obrazovek těchto mobilních telefonů. Zařízení, která jsem při testování mé aplikace použil, jsou zachycena v následující tabulce 3.

Tabulka 3: Seznam použitých mobilních telefonů k testování aplikace

Mobilní telefon	**Verze OS Android**	**Rozlišení displeje**
LG Nexus 5	Android 6.0.1	1920×1080
Lenovo A6000	Android 5.0.2	1280×720
Lenovo A6000	Android 4.4.4	1280×720
Lenovo TAB 2 A7-30 3G	Android 5.0.1	1024×600
Samsung Galaxy Core Duos	Android 4.1.2	800×480

Jako referenční telefon byl využit můj LG Nexus 5, na kterém byla aplikace vyvíjena a postupně odlaďována. Při testování jsem na tomto telefonu neobjevil žádné závažnější chyby, které by ohrožovaly korektní chod aplikace. Nicméně ani na dalších testovaných přístrojích jsem nezaznamenal žádné chyby a aplikace taktéž fungovala bez vážnějších chyb.

Dále bylo nutné také otestovat serverovou stranu celého systému. Zde je již problematika testování daleko jednodušší než u aplikace pro OS Android. Metodika testování spočívala pouze v odzkoušení správnosti funkcí systému, rychlosti odezvy tohoto systému a jeho stability. Testování probíhalo ve třech různých prohlížečích (*Internet Explorer 11.0*, *Mozilla Firefox 44.0* a *Google Chrome 50.0*), přičemž ve všech proběhlo korektní zobrazení stránek a také fungovaly veškeré funkce s těmito stránkami spojené. Nakonec jsem také musel provést testy bezpečnosti této serverové komponenty. Jelikož se heslo uživatele do databáze ukládá ve formě hašovací funkce SHA-1 a nahrávání dat na server probíhá zabezpečeným přenosem pomocí protokolu HTTPS, nepředpokládám tedy nějaké narušení bezpečnosti této aplikace.

8.1 Problémy aplikace

Po otestování serverové komponenty bylo tedy vše v pořádku a na žádné problémy jsem nenarazil. Nicméně s klientskou aplikací pro OS Android jsou bohužel spojeny i problémy, které vznikaly při samotné implementaci, eventuálně jsou způsobeny nedokonalostí dnešních mobilních zařízení. O těchto problémech pojednávají následující body.

- **Natočení zařízení**

  V předchozí kapitole bylo řečeno, že natočení přístroje vůči severu je řešeno virtuálním senzorem s názvem `ROTATION_VECTOR`, který sbírá data ze tří různých senzorů (*magnetometr*, *akcelerometr* a *gyroskop*). Pomocí těchto dat se následně spočte výsledné natočení zařízení k magnetickému severu Země. Akcelerometr měří gravitační zrychlení zařízení a gyroskop slouží k určení naklonění a natočení tohoto telefonu. Pomocí těchto dvou senzorů lze tedy určit skutečný pohyb a natočení zařízení v inerciálním prostoru. Pro účely aplikace ovšem potřebujeme zjistit natočení vůči magnetickému poli Země. Toto zjistíme přidáním dat z magnetometru, což je senzor měřící magnetické pole ve všech třech osách. Tento senzor je tvořen feromagnetickým kouskem plechu, který je zmagnetizován a nazývá se magnetorezistivní senzor. V důsledku změn magnetického pole okolí, je tento senzor ovlivněn a v důsledku toho snižuje nebo zvyšuje odpor feromagnetického materiálu. Z toho tedy vyplývá, že na senzor magnetometru má vliv především výskyt elektromagnetických polí, ale i výskyt permanentních magnetů, či kovových materiálů v okolí. Právě v tomto tkví problém určení natočení zařízení. Pokud bychom telefon umístili za čelní sklo do automobilu za účelem automatického focení zájmových bodů dle polohy, vystavíme se problému, že automobil slouží do určité míry jako Faradayova klec. Automobil tedy nepropustí všechny elektromagnetické vlny a tudíž senzor není schopen přesně rozhodnout na kterou stranu je mobilní telefon natočen.

  Při testování tohoto problému jsem zjistil, že z výše uvedeného důvodu se hodnota natočení může lišit až o 90°. Toto lze do určité míry omezit kalibrací senzoru tak, že přístrojem budeme pohybovat ve tvaru čísla 8. Tento problém je tedy dán především výskytem kovových materiálů v okolí zařízení, ale také nedokonalostí technologie tohoto senzoru. Z tohoto důvodu je tedy implementována možnost vypnutí focení pouze v úhlu zobrazení.

- **Spotřeba baterie**

  Jedním z aspektů testování aplikace bylo také zhodnocení spotřeby baterie telefonu. Tento test jsem prováděl pouze na telefonu LG Nexus 5, jelikož jde o testování z dlouhodobého hlediska a testování spotřeby na více zařízeních poněkud ztrácí smysl, jelikož i tyto hodnoty by byly velmi podobné hodnotám naměřeným. K testování spotřeby a výdrže baterie byl použit program *Battery Monitor Widget* ve verzi 3.15.1. Testování probíhalo tak, že jsem náhodně přidal body na mapě a během následující půlhodiny jsem těmito body projížděl automobilem. Během této doby bylo zapnuto logování, tedy i GPS, a pokud bylo zařízení v dosahu zájmového bodu, začalo automatické focení. Za uplynulý čas byla spočtena průměrná spotřeba 859 mA. Nicméně při automatickém focení zájmového bodu se tento údaj vyšplhal dokonce na hodnotu 1255 mA. Na přístroji LG Nexus 5 s kapacitou baterie 2300 mAh by tedy teoreticky vydržela baterie v provozu 160 minut. Reálně by tato hodnota byla ovšem nižší. Z toho tedy plyne, že aplikace je energeticky velmi náročná.

To je dáno především tím, že pokud je zařízení v rádiusu 250 metrů od zájmového bodu, hledá se poloha zařízení neustále pomocí GPS, přičemž tato technologie je velmi náročná na spotřebu energie. Poloha dle GPS je ale v aplikaci velmi důležitá, jelikož potřebujeme zjistit co nejrychleji nejpřesnější polohu pro focení zájmového bodu. Dále je v aplikaci náročné na spotřebu energie také využití integrované kamery zařízení, jelikož je zobrazován reálný náhled výstupu kamery a při automatickém focení jsou fotografie foceny ihned za sebou jak jen to je možné. Nicméně toto nelze považovat za velký problém, jelikož se při použití funkce logování počítá s tím, že telefon bude umístěn v autě, kde se naskýtá možnost připojení telefonu ke zdroji napájení.

Na spotřebu baterie má samozřejmě vliv i algoritmus BIQI, který je výpočetně velmi složitý. Testování tohoto algoritmu je popsáno v následující poslední podkapitole.

8.2 Testování algoritmu BIQI

Na samotný konec testování jsem musel zkontrolovat správnost implementace algoritmu BIQI, ale také jeho časovou a paměťovou složitost. Nejprve jsem tedy porovnával výsledky mnou implementovaného algoritmu s výsledky ukázkového algoritmu napsaného v prostředí *Matlab*. Algoritmy jsem spouštěl se stejnými vstupy (obrázky) a získal jsem vždy stejné výstupy (indexy kvality). Algoritmus je tedy z hlediska funkčnosti naprogramován dobře. Následovalo zjistit časovou složitost tohoto algoritmu. Testování probíhalo vždy se stejným obrazem, ale pokaždé s jiným rozlišením. Tento obraz byl předán algoritmu a zjišťovalo se, za jakou dobu vypočte index kvality obrazu. Tento test byl pro každé rozlišení prováděn 10×, a následně byla vypočtena průměrná hodnota doby výpočtu tohoto indexu. Algoritmus byl testován na telefonu LG Nexus 5, proto se mohou hodnoty na různých telefonech lišit.

Tabulka 4: Rychlost výpočtu algoritmu BIQI v závislosti na rozlišení obrazu

Rozlišení obrazu	**Počet obrazových bodů**	**Průměrný čas výpočtu (s)**
320 × 240	76 800	1,112
640 × 480	307 200	2,172
800 × 480	384 000	3,105
800 × 600	480 000	4,076
1024 × 768	786 432	7,757
1280 × 720	921 600	8,763
1280 × 800	1 228 800	11,613
1600 × 1200	1 920 000	19,486
1920 × 1080	2 073 600	20,807

Z tabulky 4 lze tedy usoudit, že se vzrůstajícím rozlišením obrazu, roste také čas potřebný pro výpočet indexu kvality. Platí tedy přímá úměrnost a můžeme říci, že asymptotická složitost je lineární, tedy: $\mathcal{O}(n)$. Totéž platí i u paměťové složitosti, kdy se vzrůstajícím rozlišením, rostou požadavky na paměť mobilního zařízení.

9 Závěr

Cílem této diplomové práce bylo především naimplementovat aplikaci pro operační systém Android, která automaticky pořizuje fotografie na základě polohy, nebo zatřesením mobilního zařízení. Cílem této klientské aplikace je vyfotit daný počet snímků zájmového bodu a pomocí algoritmu pro hodnocení kvality obrazu vybrat nejlepší snímek ze série pořízených. Dalším úkolem práce bylo také naimplementovat webový server, kde by klientská aplikace zasílala tento nejlepší vybraný snímek spolu s informacemi, kde a kdy byl tento snímek pořízen apod. Toto spojení při zasílání fotografie mělo být zabezpečeno. V rozhraní webového serveru by si uživatel měl moci přidat nové zájmové body, které by si stahovala klientská aplikace a taktéž by zde měla být galerie na prohlížení pořízených snímků zaslaných klientskou aplikací.

Na začátku práce uvádím stručně potřebnou teorii. Poté jsou vysvětleny některé metody subjektivních hodnocení kvality obrazu a následně jsou shrnuty algoritmické možnosti objektivních hodnocení. Dále je stručně popsán vybraný algoritmus BIQI, který byl srovnáván s ostatními algoritmy a jsou uvedeny důvody výběru. Následuje návrh aplikace a dále kapitola o její implementaci. Na samotný závěr je uvedeno testování aplikace a možné problémy.

Tato diplomová práce mi byla velikým přínosem, jelikož jsem si osvojil některé nové principy programování a naučil jsem se novým metodám a technikám vývoje softwaru. Především jsem se ale něco dozvěděl o algoritmických metodách objektivního hodnocení kvality obrazu, což je stále rozvíjející se obor v multimediální technice.

Téma práce jsem si vybral, jelikož pro mě bylo velmi zajímavé a také proto, že neznám, ani jsem nevěděl o obdobné aplikaci, která by fungovala na tomto nebo podobném principu. Při vývoji této aplikace jsem zjistil proč tomu tak je. Lidé jednoduše takovouto funkcionalitu nepotřebují, nebo nemají důvod ji použít a to ze dvou důvodů. Prvním je již zmíněná nedokonalost technologie senzorů v mobilních zařízeních a s tím spojená nemožnost přesně určit zájmový bod, který chceme fotit. Druhý důvod je ale daleko podstatnější, a to, že tuto problematiku již dávno řeší 360°kamery umístěné na automobilech, kolech, batozích apod. Tyto kamery poskytují celý úhel záběru daného místa a tudíž není problém vyfotit a určit daný zájmový bod. Toto reálné mapování je již na různých serverech vyřešeno a z tohoto důvodu není má aplikace pro reálné využití příliš užitečná. Nicméně tématika hodnocení kvality obrazu by mohla být v řadě aplikací užitečná, a tedy s patrnými úpravami mé aplikace, by byla pro uživatele jistě zajímavá.

Aplikaci prozatím nehodlám nijak publikovat. Jednak z výše zmíněného důvodu, a jelikož by pravděpodobně náklady na provoz serveru převyšovaly zisk aplikace. Ovšem pokud bych v budoucnu aplikaci distribuoval na *Google Play*, patrně bych musel postupně přidávat nové funkce na základě ohlasů uživatelů. Mezi takové funkce by např. mohlo patřit: focení zájmových bodů i s bleskem, vylepšení implementace odchycení otřesu zařízení, mazání jednotlivých bodů na serveru a další různá nastavení.

Literatura

[1] ROUSEK, Marek. *Objektivní hodnocení kvality obrazu mluvčího znakového jazyka*. Praha, 2016. Bakalářská práce. České vysoké učení technické v Praze. Vedoucí práce Ing. Martin Bernas, CSc.

[2] WANG, Zhou a Alan Conrad BOVIK. *Modern image quality assessment*. San Rafael: Morgan, 2006, s. 17-26. ISBN 1598290223.

[3] NETUŠIL, Kamil. *Subjektivní a objektivní hodnocení kvality obrazu a videa*. Plzěň, 2011. Bakalářská práce. Západočeská univerzita v Plzni. Vedoucí práce Ing. Ivo Veřtát.

[4] *Recommendation ITU-R BT.500-13: Methodology for the subjective assessment of the quality of television pictures* [online]. Geneva, 2012, **2012**(01) [cit. 2016-03-01]. Dostupné z: `www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.500-13-201201-I!!PDF-E.pdf`

[5] Subjective quality assessment. In: *MULTIMEDIA SIGNAL PROCESSING GROUP MMSPG* [online]. [cit. 2016-03-01]. Dostupné z: `http://mmspg.epfl.ch/page-58333-en.html`

[6] NOGHE, Petr. *Objektivní hodnocení kvality videa v prostředí Matlab*. Brno, 2011. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Ladislav Polák.

[7] MOHAMMADI, Pedram, Abbas EBRAHIMI-MOGHADAM a Shahram SHIRANI. *Subjective and Objective Quality Assessment of Image: A Survey* [online]. 2014 [cit. 2016-03-01]. Dostupné z: `http://arxiv.org/ftp/arxiv/papers/1406/1406.7799.pdf`

[8] SATSANGI, Parul, Sagar TANDON, Prashant Kr. YADAV a Priyal DIWAKAR. *No-Refrence Image Quality Assessment Using Blind Image Quality Indices* [online]. Moradabad, 2013 [cit. 2016-03-02]. ISSN 2278-8948. Dostupné z: `http://www.irdindia.in/journal_ijaeee/pdf/vol2_iss2/6.pdf`

[9] MOORTHY, Anush Krishna a Alan Conrad BOVIK. *A Two-Step Framework for Constructing Blind Image Quality Indices* [online]. 2010 [cit. 2016-03-02]. ISSN 1070-9908. Dostupné z: `http://live.ece.utexas.edu/publications/2010/akm_spl_may10.pdf`

[10] WANG, Zhou, Hamid R. SHEIKH a Alan C. BOVIK. *No-Reference Perceptual Quality Assessment of JPEG Compressed Images* [online]. [cit. 2016-03-04]. Dostupné z: `http://live.ece.utexas.edu/publications/2002/zw_icip_2002_norefjpeg.pdf`

[11] ISLAM, Sheikh Md. Rabiul, Xu HUANG a Kim LE. *Mr Image Compression Based od Selection of Mother Wavelet and Lifting Based Wavelet* [online]. 2014 [cit. 2016-03-04]. Dostupné z: `http://airccse.org/journal/jma/6214ijma06.pdf`

[12] SHEIKHH, B.SC., M.S., Hamid Rahim. *Image Quality Assessment Using Natural Scene Statistics* [online]. 2004 [cit. 2016-03-02]. Dostupné z: `http://www.lib.utexas.edu/etd/d/2004/sheikhhr042/sheikhhr042.pdf`. THE UNIVERSITY OF TEXAS AT AUSTIN.

[13] MITTAL, Anish, Rajiv SOUNDARARAJAN a Alan C. BOVIK. *Making a 'Completely Blind' Image Quality Analyzer* [online]. 2013 [cit. 2016-03-04]. Dostupné z: `http://live.ece.utexas.edu/research/quality/niqe_spl.pdf`

[14] MITTAL, Anish, Anush Krishna MOORTHY a Alan Conrad BOVIK. *No-Reference Image Quality Assessment in the Spatial Domain* [online]. 2012 [cit. 2016-03-04]. Dostupné z: `http://live.ece.utexas.edu/publications/2012/TIP%20BRISQUE.pdf`

[15] MOORTHY, Anush Krishna a Alan Conrad BOVIK. *Blind Image Quality Assessment: From Natural Scene Statistics to Perceptual Quality* [online]. 2011 [cit. 2016-03-04]. Dostupné z: `http://live.ece.utexas.edu/publications/2011/moorthy_tip_2011.pdf`

[16] Android Developers. *Android Developers* [online]. [cit. 2016-02-01]. Dostupné z: `http://developer.android.com/index.html`

[17] LACKO, Ľuboslav. *Vývoj aplikací pro Android*. 1. vyd. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

[18] Google Maps JavaScript API. *Google Maps JavaScript API* [online]. [cit. 2016-02-01]. Dostupné z: `https://developers.google.com/maps/documentation/javascript/`

A Přílohy na CD

Příloha 1

Text této diplomové práce v elektronické podobě.

Příloha 2

Snímky pro subjektivní a objektivní testování ve formátu *jpg*.

Příloha 3

Hodnoty subjektivních a objektivních měření ve formátu *xlsx*.

Příloha 4

Zdrojové kódy klientské aplikace pro Android.

Příloha 5

Spustitelná klientská aplikace pro Android.

Příloha 6

Zdrojové kódy pro webový server.